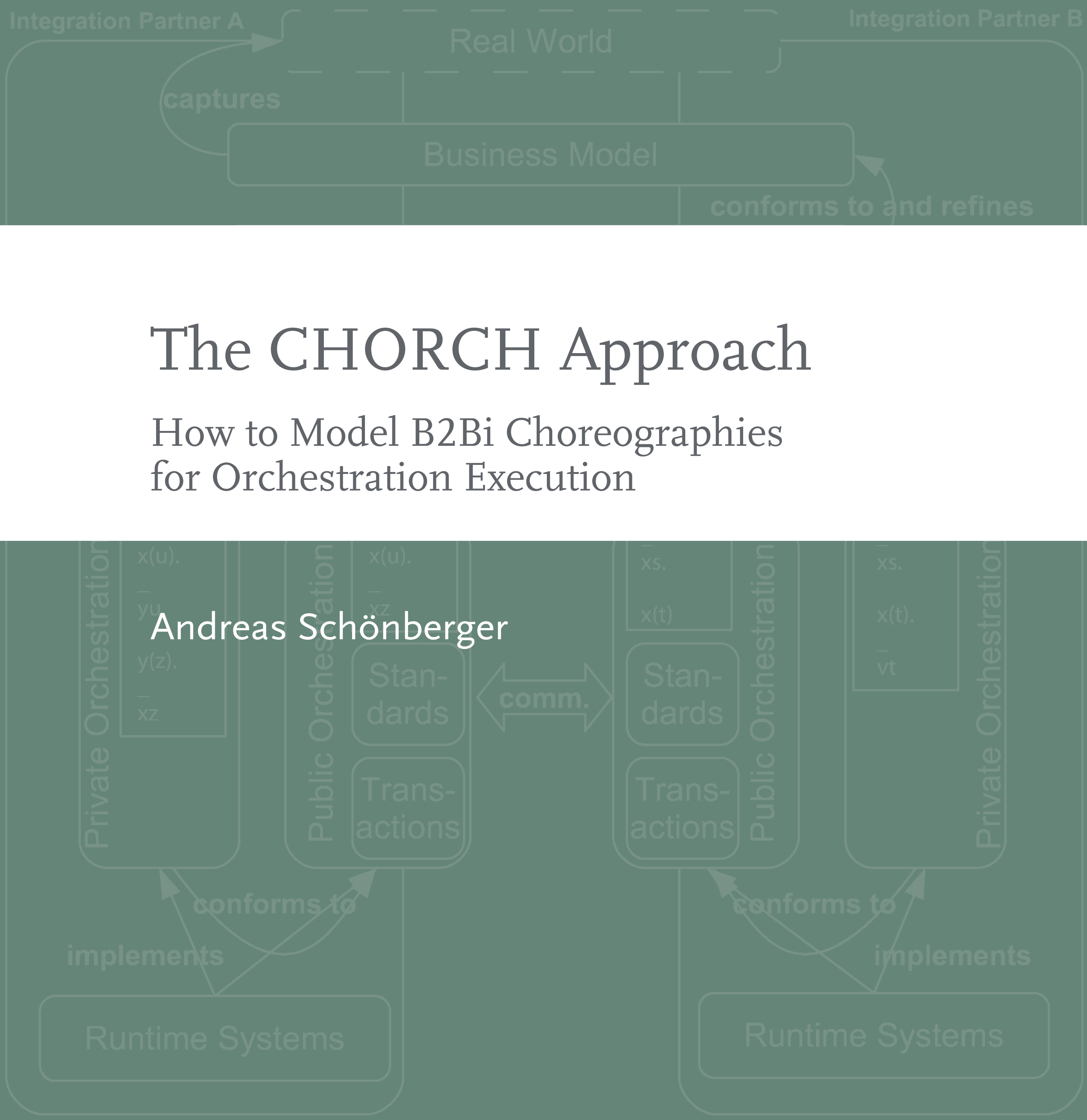
12 | Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg



The CHORCH Approach

How to Model B2Bi Choreographies for Orchestration Execution

Andreas Schönberger

UNIVERSITY OF BAMBERG PRESS

Schriften aus der Fakultät
Wirtschaftsinformatik und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Schriften aus der Fakultät
Wirtschaftsinformatik und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Band 12

University of Bamberg Press 2012

The CHORCH Approach: How to Model B2Bi Choreographies for Orchestration Execution

Andreas Schönberger

University of Bamberg Press 2012

Bibliographische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Informationen sind im Internet über http://dnb.ddb.de/ abrufbar

Diese Arbeit hat der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg als Dissertation vorgelegen
1. Gutachter: Prof. Dr. Guido Wirtz, Otto-Friedrich-Universität Bamberg
2. Gutachter: Prof. Dr. Christian Huemer, Technische Universität Wien
Tag der mündlichen Prüfung: 22. März 2012

Herstellung und Druck: docupoint GmbH, Barleben
Umschlaggestaltung: Dezernat Kommunikation und Alumni der Otto-Friedrich-Universität Bamberg

ISSN: 1867-7401
ISBN: 978-3-86309-076-0 (Druckausgabe)
eISBN: 978-3-86309-077-7 (Online-Ausgabe)
URN: urn:nbn:de:bvb:473-opus4-3924

To Vera and Vinzenz

Acknowledgments

This thesis presents the results of my research as a member of the Distributed Systems Group at the University of Bamberg. I am indebted to numerous people who directly or indirectly have contributed to the results of my dissertation project.

First of all, I would like to thank my doctoral father Prof. Dr. Guido Wirtz for giving me perfect freedom in choosing my research directions, for numerous discussions and timely feedback, for supporting my publication efforts and for insights into academic and non-academic life. I would also like to address the other members of my doctoral thesis committee, Prof. Michael Mendler, PhD, and Prof. Dr. Tim Weitzel, who offered support and shared interest in my research topic.

In addition, I would like to thank all the students who significantly contributed to particular aspects of the research topic or just inspired improvements through critical questions. Outstanding contributions were provided by the following students which also resulted in joint scientific publications:
Christoph Pflügler and his translation of ebBP-ST into BPEL, Simon Harrer and his model-driven approach for mapping ebBP BusinessTransactions to BPEL orchestrations, Johannes Schwalb and his examination of WS-* interoperability, Matthias Geiger and his SPIN validation of the ebBP BusinessTransaction execution model as well as Jörg Lenhard and his evaluation of the expressiveness of orchestration languages. Joint student results were produced during a software development project in the summer of 2008, in particular by Johannes Schwalb, Matthias Geiger, Simon Harrer and Thomas Benker. All student contributions are explicitly marked throughout this thesis.

Furthermore, I would like to thank all my former and current colleagues for collaboration. Sven Kaffille and Dr. Christof Simons in particular for introducing me to the Distributed Systems Group, for motivation and friendship. Matthias Geiger, Simon Harrer, Jörg Lenhard and Christian Wilms for support while I was writing the thesis. Mostafa Madiesh, Stephan Scheele, Philipp Eittenberger, Dr. Daniel Beimborn, Nils Joachim and Prof. Dr. Udo Krieger for discussions and motivation. Special thanks go to Cornelia Schecher and Babette Schaible for all the organizational support and for listening.

Moreover, I am particularly indebted to the members of the RosettaNet community for providing me with use cases and real-life validation of my proposals. I especially would like to thank Hussam El-Leithy, Nikola Stojanovic, and Dale Moberg for listening to my proposals and for exchanging ideas. Special thanks also go to Debra Praznik for organizational support and motivation. I also would like to thank all other members of the RosettaNet MCC team for discussions and collaboration.

I am deeply grateful to all national and international colleagues who reviewed,

discussed and suggested improvements to my work. Special thanks go to Prof. Dr. Christian Huemer and Dr. Marco Zapletal of the Vienna University of Technology, Dr. Andreas Wombacher of the University of Twente and Oliver Kopp of the University of Stuttgart for inspiring discussions.

Deepest thanks go to my family without whose support this thesis would not be finished by now. Vera for patience and love. Vinzenz for lighting up my eyes every day. My parents-in-law for tremendous support, in particular since Vinzenz was born. My parents and family for always believing in me and motivation.

Kurzfassung

Betriebsübergreifende Geschäftsprozessintegration ist eine logische Konsequenz allgegenwärtigen Wettbewerbsdrucks. In diesem Kontext fokussiert Business-To-Business integration (B2Bi) auf die Informationsaustausche zwischen Unternehmen. Eine B2Bi-Kernanforderung ist die Bereitstellung adäquater Modelle zur Spezifikation der Nachrichtenaustausche zwischen Integrationspartnern. Diese werden im Rahmen dieser Arbeit in Anlehnung an Service-orientierte Architekturen (SOA)-Terminologie *Choreographien* genannt. Bestehende Choreographiesprachen decken die Anforderungen an B2Bi-Choreographien nicht vollständig ab. Dedizierte B2Bi-Choreographiestandards definieren inkonsistente Austauschprozeduren für grundlegende Interaktionen und nur unvollständige Semantiken für fortgeschrittene Interaktionen. Formale oder Technik-getriebene Choreographiesprachen bieten die benötigte Präzision, lassen aber Domänenkonzepte vermissen oder operieren auf einer niedrigen Abstraktionsebene. Angesichts solcher Mängel wird die Spezifikation valider und vollständiger B2Bi-Choreographien zu einer echten Herausforderung. Gleichzeitig sind mangelhafte Choreographiemodelle gerade im B2Bi-Bereich besonders problematisch, da diese nicht nur zwischen Fach- und IT-Abteilung, sondern auch über Unternehmensgrenzen hinweg eingesetzt werden.

Der CHORCH-Ansatz schafft an dieser Stelle mittels maßgeschneiderter Choreographien Abhilfe, welche die Vorteile von B2Bi-Choreographien und von formalen Ansätzen kombinieren. Als Ausgangspunkt wird das ebXML Business Process Specification Schema (ebBP) verwendet, das als B2Bi-Choreographiestandard Domänenkonzepte wie zum Beispiel sogenannte BusinessTransactions (BTs) bietet. Eine BT ist der Basisbaustein von B2Bi-Choreographien und spezifiziert den Austausch eines Geschäftsdokuments sowie eines optionalen Antwortdokuments. Darüber hinaus bietet ebBP ein Format zur Spezifikation von BT-Kompositionen zur Unterstützung komplexer Interaktionen.

CHORCH erweitert ebBP wie folgt. Erstens, das Ausführungsmodell für BTs wird neu definiert, um inkonsistente Ergebniszustände zu eliminieren. Zweitens, für Entwicklungsprojekte werden zwei binäre Choreographieklassen definiert, die als *B2Bi-Implementierungskontrakt* dienen sollen. Die Formalisierung beider Klassen sowie formale operationale Semantiken gewährleisten Eindeutigkeit, während Validitätskriterien die Ausführbarkeit entsprechender Modelle mittels BPEL-basierter Orchestrationen garantieren. Drittens, zur Analyse der Synchronisationsbeziehungen komplexer B2Bi-Szenarien wird eine Multi-Party-Choreographieklasse nebst Analyseframework definiert. Wiederum wird für diese Klasse eine Formalisierung definiert, die mittels Standard-Zustandsautomatensemantik Eindeutigkeit gewährleistet. Ferner garantieren Validitätskriterien die Anwendbarkeit der definierten Analysealgorithmen.

Insgesamt bieten die Choreographieklassen des CHORCH-Ansatzes ein B2Bi-adäquates, einfaches, eindeutiges und implementierbares Modell der Nachrichtenaustausche zwischen B2Bi-Partnern. B2Bi-Adäquatheit wird durch Verwendung von B2Bi-Domänenkonzepten, Abstraktion von rein technischen Kommunikationsdetails und Abdeckung der meisten praktisch relevanten B2Bi-Szenarien gewährleistet. Einfachheit ist ein Ausfluss der Verwendung eines *Zustandsmaschinen*-basierten Modellierungsparadigmas, das die Definition des Interaktionsfortschritts in Form von *Zuständen* sowie einfache Kontrollflussstrukturen ermöglicht. Eindeutigkeit wird durch die Verwendung formaler Semantiken garantiert, während Implementierbarkeit (für die beiden binären Choreographieklassen) durch Angabe von Mapping-Regeln auf BPEL-Orchestrationen sichergestellt wird.

Die Validierung der CHORCH-Choreographieklassen erfolgt in zweierlei Hinsicht.

Erstens, die Implementierbarkeit der binären Choreographieklassen mit Hilfe von Web Services und BPEL wird durch die Definition entsprechender Mappingregeln belegt. Weiterhin wird das Analyseframework der Multi-Party-Choreographieklasse als Java-Prototyp implementiert.

Zweitens, für alle Choreographieklassen wird eine abstrakte Visualisierung auf BPMN-Basis definiert, die von diversen technischen Parametern des ebBP-Formats abstrahiert. Damit wird die Integrierbarkeit der CHORCH-Choreographieklassen in Entwicklungsansätze, die ein visuelles Modell als Ausgangspunkt vorsehen, belegt. CHORCH definiert, wie sogenannte *BPMN-Choreographien* zum Zweck der B2Bi-Choreographiemodellierung zu verwenden sind und übersetzt die Validitätskriterien der CHORCH-Choreographieklassen in einfache Modell-Kompositionsregeln.

In seiner Gesamtheit bietet CHORCH somit einen Ansatz, mit Hilfe dessen B2Bi-Partner zunächst die Typen und zulässigen Reihenfolgen ihrer Geschäftsdokumentaustausche auf Basis eines abstrakten visuellen BPMN-Modells identifizieren können. Im Fall von Multi-Party-Choreographien steht dann ein Framework zur Analyse der Synchronisationsbeziehungen zwischen den Integrationspartnern zur Verfügung. Im Fall von binären Choreographien können ebBP-Verfeinerungen abgeleitet werden, welche die Modelle um technische Parameter anreichern, die zur Ableitung einer Implementierung benötigt werden. Diese ebBP-Modelle sind in Web Services- und BPEL-basierte Implementierungen übersetzbar.

Damit erlaubt CHORCH die schrittweise Überbrückung der semantischen Lücke zwischen der Informationsaustauschperspektive von Geschäftsprozessmodellen und den zugehörigen Implementierungen.

Ein beachtenswerter Aspekt des CHORCH-Ansatzes ist die Verwendung einschlägiger internationaler Standards auf allen Abstraktionsebenen, im Einzelnen BPMN, ebBP, Web Services und BPEL. Die Verwendung von Standards trägt dem heterogenen Umfeld von B2Bi-Szenarien Rechnung. Zusätzlich wurden Kernergebnisse des CHORCH-Ansatzes als internationale Standards der RosettaNet-B2Bi-Community veröffentlicht. Dies belegt die praktische Relevanz des Ansatzes und fördert die Verbreitung innerhalb der B2Bi-Community.

Abstract

The establishment and implementation of cross-organizational business processes is an implication of today's market pressure for efficiency gains. In this context, Business-To-Business integration (B2Bi) focuses on the information integration aspects of business processes. A core task of B2Bi is providing adequate models that capture the message exchanges between integration partners. Following the terminology used in the SOA domain, such models will be called *choreographies* in the context of this work. Despite the enormous economic importance of B2Bi, existing choreography languages fall short of fulfilling all relevant requirements of B2Bi scenarios. Dedicated B2Bi choreography standards allow for inconsistent outcomes of basic interactions and do not provide unambiguous semantics for advanced interaction models. In contrast to this, more formal or technical choreography languages may provide unambiguous modeling semantics, but do not offer B2Bi domain concepts or an adequate level of abstraction. Defining valid and complete B2Bi choreography models becomes a challenging task in the face of these shortcomings. At the same time, invalid or underspecified choreography definitions are particularly costly considering the organizational setting of B2Bi scenarios. Models are not only needed to bridge the typical gap between business and IT, but also as negotiation means among the business users of the integration partners on the one hand and among the IT experts of the integration partners on the other. Misunderstandings between any two negotiation partners potentially affect the agreements between all other negotiation partners.

The CHORCH approach offers tailored support for B2Bi by combining the strengths of both dedicated B2Bi standards and formal rigor. As choreography specification format, the ebXML Business Process Specification Schema (ebBP) standard is used. ebBP provides dedicated B2Bi domain concepts such as so-called *BusinessTransactions (BTs)* that abstractly specify the exchange of a request business document and an optional response business document. In addition, ebBP provides a format for specifying the sequence of BT executions for capturing complex interaction scenarios.

CHORCH improves the offering of ebBP in several ways. Firstly, the execution model of BTs which allows for inconsistent outcomes among the integration partners is redefined such that only consistent outcomes are possible. Secondly, two binary choreography styles are defined as *B2Bi implementation contract format* in order to streamline implementation projects. Both choreography styles are formalized and provided with a formal execution semantics for ensuring unambiguity. In addition, validity criteria are defined that ensure implementability using BPEL-based orchestrations. Thirdly, the analysis of the synchronization dependencies of complex B2Bi scenarios is supported by means of a multi-party choreography style combined with an analysis framework. This choreography style also is formalized and standard state

machine semantics are reused in order to ensure unambiguity. Moreover, validity criteria are defined that allow for analyzing corresponding models for typical multi-party choreography issues.

Altogether, CHORCH provides choreography styles that are B2Bi adequate, simple, unambiguous, and implementable. The choreography styles are B2Bi adequate in providing B2Bi domain concepts, in abstracting from low-level implementation details and in covering the majority of real-world B2Bi scenarios. Simplicity is fostered by using *state machines* as underlying specification paradigm. This allows for thinking in the *states* of a B2Bi scenario and for simple control flow structures. Unambiguity is provided by formal execution semantics whereas implementability (for the binary choreography styles) is ensured by providing mapping rules to BPEL-based implementations.

The validation of CHORCH's choreography styles is performed in a twofold way.

Firstly, the implementation of the binary choreography styles based on Web Services and BPEL technology is demonstrated which proves implementability using relatively low-cost technologies. Moreover, the analysis algorithms for the multi-party choreography styles are validated using a Java-based prototype.

Secondly, an abstract visualization of the choreography styles based on BPMN is provided that abstracts from the technicalities of the ebBP standard. This proves the amenability of CHORCH to development methods that start out with visual models. CHORCH defines how to use BPMN choreographies for the purpose of B2Bi choreography modeling and translates the formal rules for choreography validity into simple composition rules that demonstrate valid ways of connecting the respective modeling constructs.

In summary, CHORCH allows integration partners to start out with a high-level visual model of their interactions in BPMN that identifies the types and sequences of the *BusinessTransactions* to be used. For multi-party choreographies, a framework for analyzing synchronization dependencies then is available. For binary choreographies, an ebBP refinement can be derived that fills in the technical parameters that are needed for deriving the implementation. Finally, Web Services and BPEL based implementations can be generated. Thus, CHORCH allows for stepwise closing the semantic gap between the information perspective of business process models and the corresponding implementations.

It is noteworthy that CHORCH uses international standards throughout all relevant layers, i.e., BPMN, ebBP, Web Services and BPEL, which helps in bridging the heterogeneous IT landscapes of B2Bi partners. In addition, the adoption of core CHORCH deliverables as international standards of the RosettaNet community give testament to the practical relevance and promise dissemination throughout the B2Bi community.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

AA	AcceptanceAcknowledgement, ebBP business signal type
AAE	AcceptanceAcknowledgementException, ebBP business signal type
API	Application Programming Interface
AS2	Applicability Statement 2
BA	BusinessActivity
B2B	Business-to-Business
B2Bi	Business-to-Business integration
BC	BusinessCollaboration
BCA	CollaborationActivity
BCL	Business Choreography Language
BPEL	Web Services Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Model and Notation
BSI	Business Service Interface
BSP	(WS-I's) Basic Security Profile
BSPL	Blindingly Simple Protocol Language
BT	BusinessTransaction
BTA	BusinessTransactionActivity
CCTS	Core Components Technical Specification
CGV	ConditionGuardValue, ebBP expression language
CPA	Collaboration-Protocol Agreement
CPP	Collaboration-Protocol Profile

CPPA	Collaboration-Protocol Profile and Agreement, an ebXML standard
CPS	Control Process State Machine
DOM	Document Object Model
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
EU	European Union
ebBP	ebXML Business Process Specification Schema
ebBP-Reg	ebBP Regular, a specification style for binary choreographies
ebBP-ST	ebBP Shared State, a specification style for binary choreographies
ebBP+	Extended ebBP modeling
ebMS	ebXML Messaging Services
ebRIM	ebXML Registry Information Model
ebRS	ebXML Registry Services and Protocol
ebXML	Electronic Business using XML
EJB	Enterprise JavaBeans
ESB	Enterprise Service Bus
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
GE	GeneralException, ebBP business signal type
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure (HTTP over SSL)
ICT	Information and Communication Technology
IDE	Integrated Development Environment
IOWF-Net	Interorganizational Workflow Net
IS	Information Systems
IT	Information Technology

JAXB	Java Architecture for XML Binding
JAX-WS	Java API for XML-Based Web Services
JBI	Java Business Integration
JEE	Java Platform, Enterprise Edition, also Java EE, previously J2EE
JMS	Java Message Service
JRE	Java Runtime Environment
JSE	Java Platform, Standard Edition
JVM	Java Virtual Machine
LTL	Linear Temporal Logic
LoST	Local State Transfer
MCC	Message Control and Choreography
MCM	Message Choreography Modeling Language
MDA	Model-Driven Architecture
MEP	Message Exchange Pattern
MIME	Multipurpose Internet Mail Extensions
MMS	Multiple Messaging Services
MTOM	SOAP Message Transmission Optimization Mechanism
NOF	Notification of Failure
OAGi	Open Applications Group
OASIS	Organization for the Advancement of Structured Information Standards
OFTP2	ODETTE File Transfer Protocol 2
OSGi	Open Services Gateway initiative
OSOA	Open Service Oriented Architecture
PIP	Partner Interface Process
QName	Qualified Name
QoS	Quality of Service

RA	ReceiptAcknowledgement, ebBP business signal type
RAC	ReceiptAcknowledgementCreator
RAE	ReceiptAcknowledgementException, ebBP business signal type
REST	Representational State Transfer
RIG	RosettaNet Implementation Guide
RNIF	RosettaNet Implementation Framework
RSP	(WS-I's) Reliable Secure Profile
SecRM	Secure WS-ReliableMessaging (Scenario)
SeqMP	Sequential Multi-Party, a specification style for multi-party choreographies
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	SOAP
SPIN	The SPIN model checker
SSL	Secure Sockets Layer
StAX	Streaming API for XML
SCM	Supply Chain Management
SCO	Supply Chain Orientation
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTP	TimeToPerform
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
UMM	UN/CEFACT Modeling Methodology
UN/CEFACT	United Nations' Centre for Trade Facilitation and E-business
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

VAN	Value Added Network
W3C	World Wide Web Consortium
WAS	IBM WebSphere Application Server
WSDL	Web Service Description Language
WSIT	Web Service Interoperability Technologies
WS-CDL	Web Services Choreography Description Language
WS-I	Web Services Interoperability Organization
WS-RM	WS-ReliableMessaging
WS-RM-Pol	WS-ReliableMessaging Policy
WS-Sec	WS-Security
WS-SecConv	WS-SecureConversation
WS-Sec-Pol	WS-Security Policy
WS-Stack	Web Services stack
WS-Trust	WS-Trust
XML	Extensible Markup Language
XMLNS	XML namespace
XPath	XML Path Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

1. Introduction

The integration of cross-organizational business processes, commonly referred to as Business-to-Business integration (B2Bi), is an area of enormous importance to today's economy. The sheer number of B2Bi communities such as Open Applications Group (OAGi)[1], Odette[2], papiNet[3] or RosettaNet[4], numerous business document standards [95] and large-scale B2Bi standardization projects such as ebXML[5] and UMM [210] are testament to this fact. Considering some figures of RosettaNet, a major B2Bi community which defines business document formats and exchange procedures, show that B2Bi transactions implemented using RosettaNet's Partner Interface Processes (PIPs) alone are worth billions of dollars[6]. The importance of B2Bi is also consistently reflected in academic research [1, 51, 90, 124, 231].

Unfortunately, the importance of the B2Bi domain is paired with its complexity. The task of automating B2Bi scenarios, which is central to this thesis, is challenged by both organizational and technical issues.

At the specification level, adequate support for modeling the types, sequences, and effects of business document exchanges is required. Contrary to in-house software implementation projects, this also concerns coordination between personnel from enterprises with diverse cultural backgrounds, levels of knowledge, goals and domains of control. For the good of enterprise, business experts have to agree upon the content and meaning of business document exchanges whereas software engineers have to agree upon the technical configuration of messaging facilities. Furthermore, IT has to be aligned with agreed-upon business goals. In this organizational setting, changes to either of the agreements would require extensive discussions among all implementation team members and should thus be avoided.

At the implementation level, the distributed nature of B2Bi systems plays a major role in terms of complexity. Legacy systems must be coupled, which is not only a problem of data formats and connectivity, but also a question for the processing model, for example, *batch-style* or *on demand*. Heterogeneity between the integration systems of B2Bi partners must be overcome and typical distributed computing problems such as lost, duplicated, or delayed messages, as well as partial failures must be addressed.

[1] `http://www.oagi.org/`, last access: 12/20/2011
[2] `http://www.odette.org`, last access: 12/20/2011
[3] `http://www.papinet.org/`, last access: 12/20/2011
[4] `http://www.rosettanet.org/`, last access: 12/20/2011
[5] `http://www.ebxml.org/`, last access: 12/20/2011
[6] RosettaNet, RosettaNet Standards Assessment 2008, 2008, `http://www.rosettanet.org.my/Download/2009%20ImplementationStatistics%2005.26.09.pdf`, last access: 12/20/2011

Without adequate modeling support, business experts and software engineers are likely to come up with integration designs that are not fit-for-purpose (on the chosen platform) or may lead to inconsistencies in the participating information systems in case of message transmission errors.

The traditional way of dealing with distributed computing issues in the B2Bi domain is the application of so-called Business Service Interfaces (BSIs) that govern the cross-organizational message exchanges. Figure 1.1, which is taken from the RosettaNet Implementation Framework (RNIF) documentation [170] exemplifies the use of BSIs by conceptualizing the exchange of business documents between trading partners A and B (represented as shaded rectangles). The internal processes

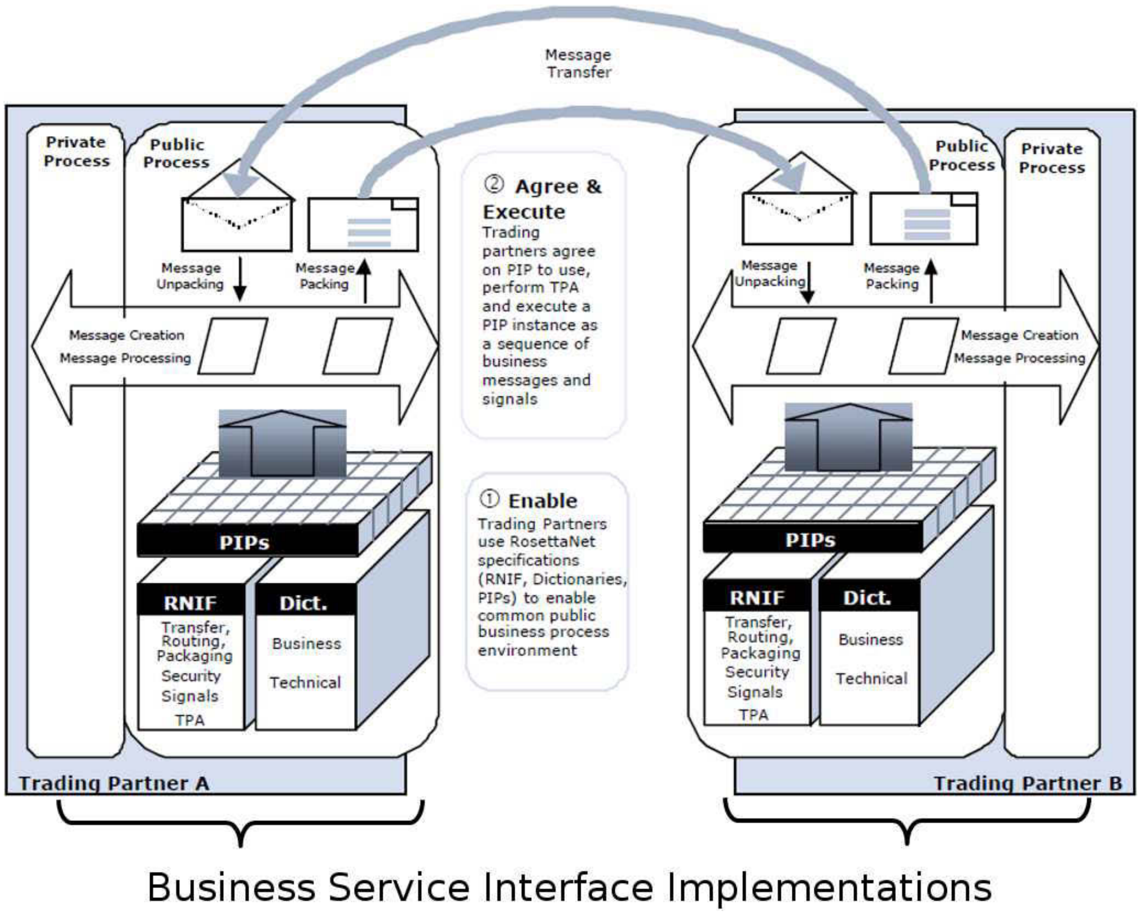

Message Transfer
Private Process
Public Process
Message Unpacking
Message Packing
Message Creation
Message Processing
② Agree & Execute
Trading partners agree on PIP to use, perform TPA and execute a PIP instance as a sequence of business messages and signals
① Enable
Trading Partners use RosettaNet specifications (RNIF, Dictionaries, PIPs) to enable common public business process environment
PIPs
RNIF
Transfer, Routing, Packaging
Security
Signals
TPA
Dict.
Business
Technical
Trading Partner A
Trading Partner B
Business Service Interface Implementations

Figure 1.1.: Business Service Interface Implementation (adapted from [170])

of the two integration partners are depicted on the outer edges of the diagram as long vertical rectangles with rounded corners and are labeled *'Private Process'*. These private processes are typically carried out by some business applications such as Enterprise Resource Planning (ERP) systems and coordinate the consumption of the respective company's resources for accomplishing its business goals. For cross-organizational communication, so-called *'Public Processes'* are used that are also depicted as long vertical rectangles with rounded corners (inner part of the figure). The tasks that are fulfilled by those public processes are manifold. Company internal data representations have to be converted into a message format that is commonly understood by both partners. Furthermore, the data needs to be packaged within message containers for transmission and these messages are to be exchanged in a secure and reliable manner. Public processes also are in charge of unpacking data and reconversion of data into internal formats. Supporting these tasks is a typical application area of B2Bi standards. The use of some RosettaNet standards is consistently represented by various symbols within the *'Public Process'* rectangles, which will be returned to later.

A BSI's task is characterized by implementing public processes and integrating these with private processes. However, integration between public and private processes can be interpreted with a high degree of variance.

At one extreme, a BSI may be used as a simple messaging proxy that offers nothing more than basic system connectivity and transport control. Internal business applications therefore simply use it to transmit single messages as self-contained and complete activities. Even the configuration of some messaging characteristics such as communication endpoints or communication qualities like security or reliability may be provided by business applications in this case. At the other extreme, a BSI takes control of the cross-organizational processes, implementing part of the business logic and steering the flow of message exchanges. Both extreme situations are undesirable from a software engineering perspective. On the one hand, implementing integration logic in business applications impedes the *separation of concerns* software engineering principle and unnecessarily ties business software to distributed systems frameworks. On the other hand, implementing part of the business logic outside of business applications leads to scattered application landscapes with intricate dependencies between system components.

In practice, the implementation and management of BSIs is frequently outsourced to dedicated B2Bi solution providers such as Axway[7], GXS[8] or Seeburger[9]. Dedicated B2Bi providers have significant expertise in overcoming issues of heterogeneity and in integrating with a large set of different business application vendors. However, as there is no common interface for coupling public processes with private processes, such an outsourcing strategy bears the risk of moving part of the business logic into the domain of B2Bi solution providers and thus losing control of cross-organizational processes. Moreover, traditional Electronic Data Interchange (EDI) concepts such as Value Added Networks (VANs) that ensure reliable, secure and auditable transmission of business documents via private networks are in widespread use. At this juncture, it is worth noting that the boundaries between VAN-offerings and fully hosted B2Bi solutions that take over responsibility for integration with business applications

[7]`http://www.axway.com/`, last access: 12/20/2011
[8]`http://www.gxs.com/`, last access: 12/20/2011
[9]`http://www.seeburger.de`, last access: 12/20/2011

are frequently blurred in practice. Furthermore, it does not make a big difference conceptually whether a VAN is used for message transmission or a B2Bi vendor's solution that implements VAN-like communication qualities on top of Internet connections. This said, a common characteristic is that these solutions tend to be costly.

These cost concerns are backed up by the Odette organization, a European automotive industry group for creating *"standards for e-business communications, engineering data exchange and logistics management"* that has carried out a survey among 607 IT responsibles and key users from the European automotive industry on *Data Exchange via Portals and EDI*[10]. In the survey, *cost* is identified as a significant barrier to B2Bi. This finding corresponds to a complementary Odette analysis on the *Comparison of File Transfer Alternatives for Business-to-Business (B2B) Data Exchanges*[11]. Cost, security, reliability, implementation, and applicability are the core criteria for this comparison. Internet-based solutions such as Applicability Statement 2 (AS2) or ODETTE File Transfer Protocol 2 (OFTP2) [66] over the Internet are judged to be:

> *"Globally available, cheap, used for multiple purposes, every company connected. Bandwidth scalable. No built in security. Not possible to apply SLA requirements end to end between partners. In practice there is a growing acceptance of Public Internet services for many automotive use-cases, with certain exceptions. Public Internet is available as a service from a large number of providers (ISPs)."* Odette, Comparison of File Transfer Alternatives for B2B Data Exchanges, 2011[11]

Beyond cost, a lack of process perspective is a frequent issue for implementing BSIs. This problem is twofold. Firstly, the exchange of business documents may not be integrated with internal business applications, which is to say, the exchange of business documents between BSI and business applications is not implemented. Secondly, the types of interactions may frequently not go beyond simple request-reply interactions.

In the previously mentioned Odette study[10], EDI is found to outperform web portals due to the improved process integration facilities of EDI. In addition, this study reveals that (except for EDI) web portals, e-mail, and even fax, the phone or mail dominate the B2Bi landscape. The latter strongly discourage the automation of process integration. The need for better process integration is also backed up by the establishment of the European Union (EU)-funded *auto-gration* project[12], which *"aims to improve the integration of SMEs in global digital automotive supply*

[10]Odette, *"Data Exchange via Portals and EDI"*, 2009, `https://forum.odette.org/publications/b2b/Data_Exchange_Survey_Analysis_2009`, last access: 12/20/2011

[11]Odette, *"Comparison of File Transfer Alternatives for B2B Data Exchanges"*, 2011, `https://forum.odette.org/publications/telecommunications/OP06_File%20Transfer%20Alternatives.pdf/at_download/file`, last access: 12/20/2011

[12]`http://www.auto-gration.eu/`, last access: 12/20/2011

chains"[13]. According to an *eBusiness adoption survey*[14] performed during the auto-gration project, *"even in the best cases 2/3 of the business relationships are still based on manual interaction"*. It is to note here that the deliverables of the Odette group or the auto-gration project goals are not targeting the provision of advanced process description formats that could be used as contractual agreements between integration partners for pinning down the set of admissible business document exchange sequences. In these studies, the term *"process integration"* is not used to refer to B2B interactions that include multiple business document exchanges and leverage non-trivial control flow expressions. Rather, the automation of message exchanges between BSIs and business applications is considered to be the paramount objective. Similar gaps can be identified for other B2Bi communities like OAGi[15], papiNet[16] or RosettaNet[17]. Note also that the *RosettaNet Methodology for Creating Choreographies* [173] has been developed during this dissertation project to a large extent and thus does not change the evaluation of the starting point for this work.

High cost and poor process integration call for the application of advanced process management technologies in combination with low-cost Internet communication technologies. The establishment of efforts such as RosettaNet's Message Control and Choreography (MCC) program[18] to which a large subset of this thesis' results have been submitted, or the UMM standard [210] of the United Nations' Centre for Trade Facilitation and E-business (UN/CEFACT) substantiate this. In addition, a large variety of process modeling techniques and languages ranging from process algebras [58, 113] and Petri nets [70], through academic approaches like BPEL4Chor [31] or Let's Dance [245], to international standards like ebBP [134] and Business Process Model and Notation (BPMN) [150] are adapted to or directly target B2Bi processes. Furthermore, academic research [1, 124, 231] explicitly postulates the need for advanced process integration standards:

> *"Another critical issue that impedes the supply chain-wide adoption of the [B2Bi] technologies is lack of stringent, vendor-independent business process standards. Without them, implementations can remain costly and incompatible, making a critical barrier to achieving the high level alignment and flexibility for agile supply chains. Although the standards are relatively stringent at the infrastructure-level such as with SOAP and WSDL in the Web Services area, Messaging Services (MS) of ebXML,*

[13]`http://www.auto-gration.eu/The%20Project`, last access: 12/20/2011

[14]auto-gration project, *"Analysis Report on e-Business adoption in the automotive sector"*, 2010, `http://www.auto-gration.eu/downloads/public-information/Analysis%20Report%20on%20eBusiness%20adoption%20in%20the%20automotive%20sector.pdf/at_download/file`, last access: 12/20/2011

[15]`http://www.oagi.org/`, last access: 12/20/2011

[16]`http://www.papinet.org/`, last access: 12/20/2011

[17]`http://www.rosettanet.org/`, last access: 12/20/2011

[18]`http://www.rosettanet.org/Standards/RosettaNetStandards/MessageControlandChoreography/tabid/3367/Default.aspx`, last access: 12/20/2011

and RosettaNet Implementation Framework (RNIF) of RosettaNet, there is a serious gap at the higher level for specific business processes that are essential to achieve the plug-and-play style interoperability" [1]

Choreography and orchestration technology [159] as defined in the area of Service-Oriented Architectures (SOAs) seems to provide immediate help in addressing the aforementioned problem. While choreographies define the types and sequences of publicly visible message exchanges between communication partners, orchestrations define the executable local processes of each partner (for exact definitions, please see section 2.3). This seems to perfectly fit the scenario of figure 1.1 by using choreographies as a contractual agreement between B2Bi partners on the message exchanges to be implemented and for providing the corresponding implementations by means of partner-local orchestrations that collaboratively realize the choreographies. This is especially true in light of the distinction between *abstract* and *executable* processes in the leading orchestration standard Web Services Business Process Execution Language (BPEL) [137]. An important application area of abstract BPEL orchestrations is the definition of the publicly visible message behavior of one interaction partner. Executable BPEL orchestrations are then used to complete the integration logic for existing systems. Thus, an abstract (or public) orchestration can be used to define the role of an interaction partner within a choreography, while an executable (or private) orchestration provides the corresponding implementation. It is however worth noting that there is a major difference in the relations between public (or abstract) and private (or executable) *orchestrations* as described in the BPEL standard, and public and private *processes* as described by RosettaNet (see figure 1.1). A private orchestration is a refinement of a non-executable public orchestration. This contrasts to a private process which represents a set of business applications that interact with an executable public process. In this respect, public and private orchestrations are not a one-to-one replacement for public and private processes. Rather, a public orchestration can be used to define the interface of a public process and a private orchestration can be used to provide its implementation.

Despite this excellent conceptual fit, the application of choreography and orchestration technology to B2Bi settings is far from straightforward. The terms choreography and orchestration were originally tied to Web Services [159] and early choreography and orchestration languages focused on Web Services technology. Admittedly, the distributed setting of Web Services based systems implies commonalities with B2Bi systems in terms of system assumptions, failure models, available algorithms and requirements. However, there are fundamental differences regarding the modeling constructs and engineering methods applied to B2Bi systems on the one hand, and service based systems on the other.

Traditional B2Bi is business document driven in the sense that business documents are used as the central objects of coordination between integration partners. Interactions are designed as sequences of business document exchanges and the exchange of business documents is accompanied by dedicated concepts such as disposition notifications or validity signals. From a business perspective, the choice of technology for performing the actual business document exchanges is of minimal importance. In practice, VANs, AS2 [116], File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP) all are popular methods for exchanging business documents with diverse effects on integration systems. B2Bi solution providers mediate these effects by providing dedicated integration solutions that support an array of transport technologies and target smooth integration with the existing IT landscape of integration partners.

Conversely, service based systems are designed for general-purpose content. The central object of coordination is the service which is described in a platform-independent manner and can be delivered using virtually any protocol. The homogenization of service delivery is a prerequisite for agile replacement and modification of services and hence a prerequisite for SOAs which promise business alignment, agile management as well as the reuse of IT resources. Standardization and interoperability are key enablers to homogeneous service delivery and hence foster the straightforward design and implementation of service compositions as choreographies and orchestrations. From a conceptual perspective, service based systems eliminate the need for B2Bi gateway providers.

These identified differences between B2Bi systems and services systems are particularly relevant for the design of choreographies that represent the agreements between integration partners on the business document exchanges to be implemented. Business experts and software engineers need models for communication and agreement that fit the domain. The choice and exact configuration of implementation technology is then a problem to be addressed thereafter. The following is an outline of the different classes of choreography languages that reflect B2Bi and services technology differences:

- **B2Bi Choreographies** that offer B2Bi specific concepts like business document based *BusinessTransactions* and which are semantically close to business process models.
- **Services Choreographies** that offer Web Services technology specific concepts and are close to orchestration models.
- **Conceptual choreographies** that offer concepts driven by the purpose of analysis and may be used to complement/analyze business process models as well as orchestration models.

The existence of these classes is discussed in [179] as well as in section 2.3 and there are reputable examples for each class (cf. section 2.3).

There are few approaches today that leverage dedicated B2Bi choreography languages like ebBP or UMM to specify unambiguous descriptions of the admissible business document exchanges between integration partners. Admittedly, approaches that rely on services choreography languages like Web Services Choreography Description Language (WS-CDL) [223] or BPEL4Chor [31] or conceptual choreography languages like Interorganizational Workflow Nets (IOWF-Nets) [214] or Let's Dance [245] also capture significant aspects of B2Bi scenarios. Some of these approaches also lend themselves better to formal analysis and are more easily integrated with services technology. However, as dedicated B2Bi concepts are not available, these have to be modeled *"by hand"* which impedes streamlined automation of B2Bi scenarios. Furthermore, to not take the typical use of BSIs for separating cross-organizational message exchanges from business applications into account, would result in a poor separation between integration logic and application logic.

While more services oriented and conceptual approaches tend to lack B2Bi domain concepts, approaches that use dedicated B2Bi choreography languages tend to neglect the formalization of process models and characterizations of control flow soundness, which is indeed the central concern for this thesis:

> This thesis sets out to support the automation of B2Bi scenarios with both B2Bi concepts for streamlining modeling and implementation as well as the formal underpinnings for ensuring control flow soundness.
>
> At the specification level, this thesis provides unambiguous formal semantics for B2Bi choreography models, the identification of classes of choreography models that are tailored to common needs of B2Bi scenarios, and rules for creating sound B2Bi choreography models.
>
> At the implementation level, this thesis proposes a clear concept for delineating integration logic and business logic and a prototypic implementation based on Web Services orchestrations that facilitates the usage of low-cost Internet connections.

The following sections will derive the scope of this undertaking more precisely (section 1.1), formulate the research question (section 1.2) and provide a review of the applied research method (section 1.3). Finally, section 1.4 outlines the structure of the thesis.

1.1. Scope of Work

The scope of this thesis is made more concrete by relating it to the larger domain of Supply Chain Management (SCM), by identifying the abstraction layer of choreography and orchestration technology and by identifying the type of B2Bi that the proposed models and algorithms of this thesis are applicable to.

1.1.1. B2Bi as SCM Component

SCM as a research discipline covers all relevant aspects of interorganizational relationships between businesses and therefore is suited to specify the scope of this thesis. According to [112] a *supply chain* is *"a set of three or more entities (organizations or individuals) directly involved in the upstream and downstream flows of products,* *services, finances, and/or information from a source to a customer."* [90] points out that a supply chain is not a purely linear structure, but rather has the form of an *"uprooted tree"*, as depicted in figure 1.2, which demonstrates that businesses are connected directly and indirectly to a complex network of supply chain members through their suppliers and customers. SCM can then be defined relative to the

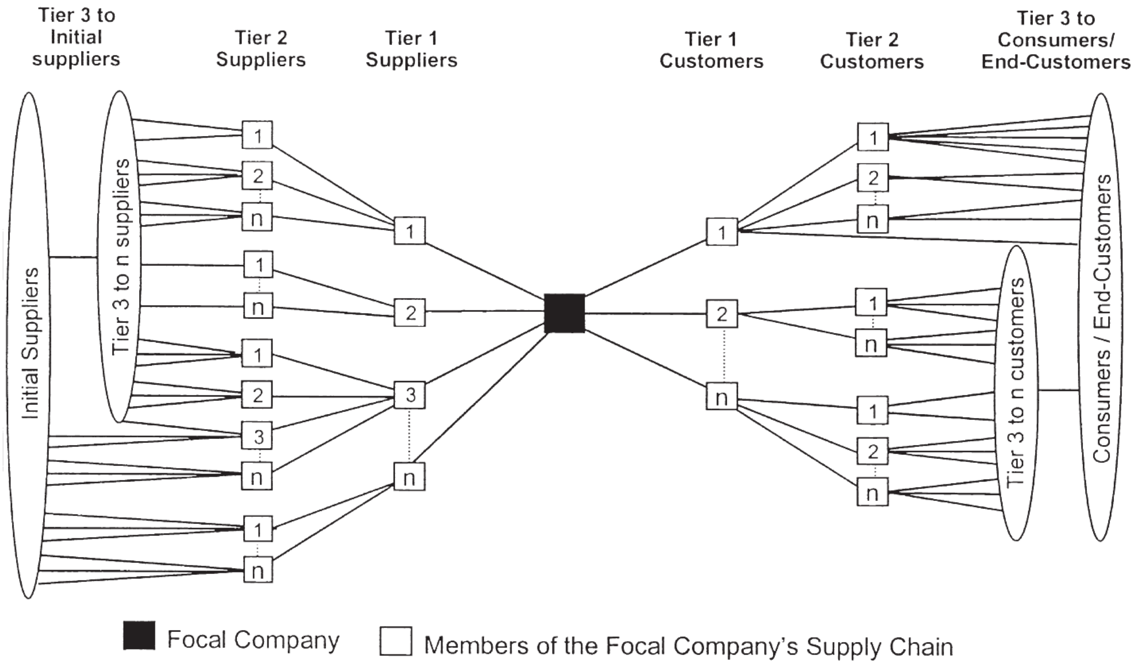


Figure 1.2.: Supply Chain Network Structure (taken from [90])

concepts of *supply chain* or *supply chain network*. [112] breaks down the notion of SCM by distinguishing the concept of Supply Chain Orientation (SCO) *"as the recognition by an organization of the systemic, strategic implications of the tactical activities involved in managing the various flows in a supply chain"* from the concept of actual SCM which *"is the implementation of a supply chain orientation across suppliers and customers"*. The implementation of SCM requires a variety of activities that go far beyond the technical implementation of message exchanges using choreography and orchestration technology. Before the active management of process links between members of a supply chain can even start, the configuration of the supply chain network structure, as well as the selection of relevant supply chain business processes, has to be performed (cf. [90]). Moreover, active management of process links necessitates several distinct management components according to [112] and [90] that range from more managerial components such as *"Risk and Reward Structure"* or *"Culture and Attitude"* to more technical components such as *"Planning and Control Methods"* or *"Communication and Information Flow Facility Structure"*.

This thesis exclusively focuses on the technical aspects of providing the communication and information flow facility structure. Managerial and organizational aspects of SCM are deliberately neglected as the technical implementation of the information flow facility structure is a challenging research topic in and of itself. [90] points out that *"virtually every author indicates that the information flow facility structure is key. The kind of information passed among channel members and the frequency of information updating has a strong influence on the efficiency of the supply chain. This may well be the first component integrated across part, or all, of the supply chain."* However, this aspect is not only an SCM management component in its own right, it is also a key enabler for several other management components. For example, current forecasts and timely monitoring information are crucial for *"Planning and Control Methods"* and the realization of *"Work Flow/Activity Structure"* is not conceivable without adequate IT support.

> For the purpose of this thesis, the term B2Bi is henceforth used to refer to the design and implementation of information exchanges between integration partners as one of the core activities of SCM.

1.1.2. Relevant Abstraction Layers

The design and implementation of IT artifacts that facilitate B2Bi are performed in phases and differ in both purpose and level of abstraction. Therefore, the identification of relevant abstraction layers that choreography and orchestration technology apply to helps in setting the scope of this thesis. The B2Bi schema depicted in figure 1.3 has been developed [187] through analyzing the abstraction levels of B2Bi projects, its development phases and the purpose of relevant B2Bi standards.

This B2Bi schema, which has evolved from the Open-edi reference model [69], begins with the *real world* level that represents the business processes to be integrated. In a perfect world, the *business model* for integration is first captured, which concerns the exchange of values between partners on an abstract level [37]. This business model is then further refined into a *business process model* that specifies all necessary activities to successfully implement the business model. This includes physical and financial activities as much as information processing activities. The next refinement step leads to the *choreography level* which focuses on the overall message exchanges of the integration partners. The differences between the latter two models are blurred, in particular if *business process models* leave out details about physical and financial activities. However, the level of technical detail of choreography models is typically significantly higher in order to be suitable for a contractual agreement regarding the admissible information exchanges. Depending on the type of choreography language, technical improvements may even comprise the exact business message schemata or communication quality attributes such as security or reliability parameters. The B2Bi schema then foresees the definition of *public orchestrations* to capture the message exchanges of the individual integration partners (for precise definitions please see section 2.2). The three small rectangles within the *public orchestration* rectangles of figure 1.3 represent major issues to address on the level of abstraction, i.e., the definition of the control flow, the selection of communication standards and the realization of transactions that keep the information systems of the integration

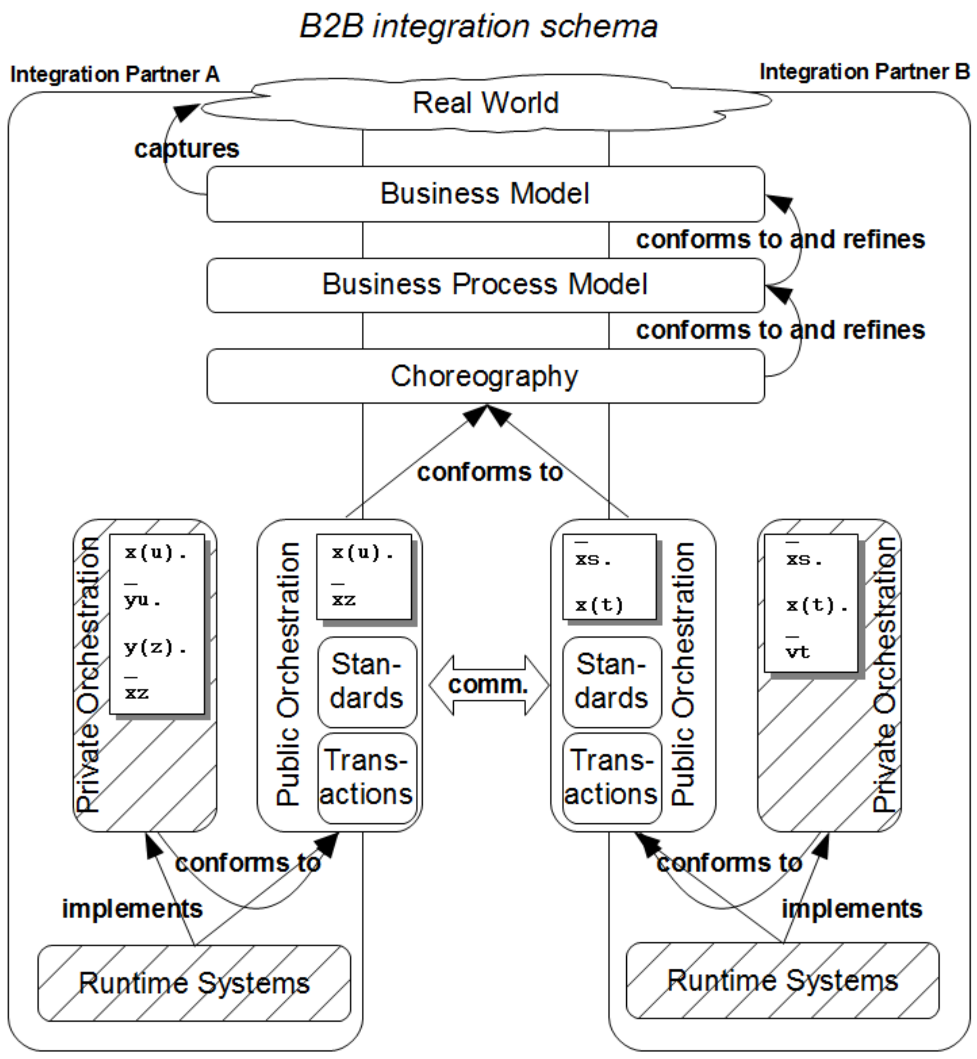


Figure 1.3.: B2Bi Schema (adapted from [187])

partners consistent. Interoperability and consistency between *public orchestration* models of the integration partners are of paramount importance. Therefore, *public orchestrations* should be interpreted as contracts that define the obligations of each integration partner. In order to simplify analysis and to hide internal details from integration partners, the B2Bi schema allows *public orchestrations* to capture only the externally observable message sequences whereas the integration with backends is to be specified in *private orchestrations*. Finally, the *runtime systems* level describes the deployment and execution of *private orchestrations*. This covers aspects such as the configuration of endpoint information for accessing backend systems, monitoring *private orchestration* instances and collecting runtime data for performance management.

This thesis seeks to address the semantic gap between *business process models* and *private orchestrations* of B2Bi scenarios by providing textual and visual models for choreography specification and showing how these can be effectively performed using orchestration technology. However, the identification and derivation of those choreography artifacts from business process models is not covered. Therefore, the implementation and management of backend systems as well as organizational aspects of managing runtime systems are not within the scope of this work.

1.1.3. Types of B2Bi

Different types of B2Bi have been identified in literature, and this section will introduce those that this study focuses on:

- [48] identifies *two basic types of B2Bi* , namely extended enterprise integration and market B2Bi. Extended enterprise integration covers the relationships of integration partners who know each other and have agreed to do business with each other for an extended period of time (cf. [48]). They know in advance that their partner can provide more or less the necessary services and thus are willing to make partner-specific IT investments. Market B2Bi on the other hand, covers relationships that do not allow for partner-specific IT investments, because, at the extreme, integration partners do not know each other in advance and potentially choose services from different partners for each transaction. Clearly the boundaries between extended enterprise integration and market B2Bi are fluid and [48] discovered that *"market B2Bi primarily concerns indirect integration through electronic marketplaces"*.

- [194] identifies and evaluates three different strategies for *"cross-organizational service composition"* which may be directly applied to B2Bi. These are *"a highly centralized solution (a central hub allows users to find potential business partners, to collaboratively model service choreographies and to finally execute them)[...,] a hybrid approach (which applies a decentralized service choreography, but is supported by a central hub) [...and] a fully decentralized, peer-to-peer architecture which works without any central entity."* Vital here is that the latter two solutions both use separate orchestration engines for each integration partner, but the hybrid approach employs a central server for, among others, managing data and process templates as well as for collaborative modeling. Finally, [194] propose an event-driven architecture based on the concept of a so-called *event bus* for integration scenarios with *"a huge degree of variability and complexity"*. The drawback of this solution is that all integration partners are forced to use the proposed *event bus*, which serves as a kind of message backbone similar to enterprise service buses.

- In [213], B2Bi is classified in accordance with the degree of human involvement, i.e., *Straight Through Processing*[19] without human involvement and *Case*

[19]The term *Straight Through Processing* actually stems from the banking sector and denotes

Handling with human involvement. The goal of straight through processing is the maximization of automation and thus the reduction of costs and cycle times. Case handling pays tribute to the fact that *"many processes are much too variable or too complex to capture in a process diagram"* [213].

Clearly, the requirements and solutions for applying choreography and orchestration technology to B2Bi scenarios may vary heavily with the B2Bi type under consideration. Case handling models the intended flow of execution of a case but, if not explicitly forbidden, also allows other flows that skip authorizations or undo activities [213]. Case handling has therefore more challenges to overcome with respect to flexibility or semantic constraint management than straight through processing. In addition, [48] identifies that the use of standards or registries such as UDDI [130] is less important for the extended enterprise integration type of B2Bi than for market B2Bi. On the contrary, however, other requirements such as security are equally important for each B2Bi type. This is not to be confused with the suitability of different architectural styles for fulfilling particular requirements. [194] compares the different identified integration strategies according to several criteria and finds that *central orchestration* lends itself better to guaranteeing security than *decentralized orchestration without a hub*.

This thesis focuses on the extended enterprise type of B2Bi by providing precise choreography and orchestration models. As the majority of the models is executable or at least has execution semantics, this corresponds to straight-through processing. As integration partners are assumed to be autonomous entities, the hybrid or fully decentralized architectures as defined by [194] are supported. However, this work does not focus on collaborative editing of choreography models, but rather identifies choreography models that can be executed using orchestrations, or at least can be validated using an analysis framework.

1.2. Research Question

This thesis is dedicated to the semantic gap between the abstraction levels of business process models and private orchestrations as identified in section 1.1.2 (figure 1.3). For the type of B2Bi systems identified, the application of process-based description models, in particular choreographies and orchestrations, is proposed. Performance and cost gains of such a process-based approach are assumed to outweigh initial setup costs. This is in line with industry efforts such as RosettaNet's MCC or the EU-funded auto-gration project (cf. above), available B2Bi standards like ebBP and UMM, and academic research [1, 51, 90, 120, 124, 231].

end-to-end automation of trading activities throughout the value chain [231]. This is different from the usage of the term in [213] where the automation of the control flow between subsequent activities of one focal role is denoted.

The use of dedicated B2Bi choreographies is proposed for several reasons. Firstly, the support of B2Bi concepts such as business documents makes this class of choreographies accessible to the B2Bi community. Secondly, B2Bi choreographies tend to be implementation technology agnostic, which is a prerequisite for being applicable to the wider set of implementation technologies used in the B2Bi domain. Thirdly, the semantic proximity to business process models on the one hand, and the message exchange focus on the other hand, promise that B2Bi choreographies are amenable to both business users and software engineers. Thus, this thesis should aid the seamless transition of business requirements into implementation systems.

In addition, service based realizations of orchestration technology are proposed due to the conceptual benefits of services technology in bridging heterogeneous platforms and coupling legacy systems.

Taking all the aforementioned into account, this project aims to answer:

RQ 1 *How can B2Bi choreographies be used as implementation contracts for services orchestration based B2Bi systems?*

This refers to the B2Bi abstraction layers identified in the last section, which provides the methodological basis for splitting the research question up into smaller and more manageable parts:

RQ 1.1 *What is a suitable B2Bi choreography language?*

This sub-question focuses on the choreography layer only. In particular, the relevant concepts needed for modeling B2Bi choreographies have to be identified and the composition of these concepts such that valid and implementable models result has to be characterized. Furthermore, an unambiguous execution semantics of choreography models is to be defined.

RQ 1.2 *To what extent and how can services orchestrations be used to implement valid B2Bi choreographies?*

This sub-question looks at the transition from the choreography layer to the orchestration layer and concerns the soundness and completeness of implementing the choreography execution semantics as well as the architectural fit of implementations with B2Bi settings.

RQ 1.3 *How can B2Bi choreographies be visualized?*

This concerns the transition from the business process model layer to the choreography layer and aims at the abstract visualization of the choreography's business document exchanges in order to support communication among business experts and software engineers within and across company boundaries. However, the details of deriving visual B2Bi choreography definitions that concentrate on the information flow from business process models that also concentrate on physical and financial flows is not considered here.

Supporting background reading on the above is given in the *Design Science* research method [50, 57]. Design science suggests the research problem to *"be defined as the differences between a goal state and the current state of a system"* [57]. The following lists give an overview of the current and the goal state of B2Bi. What is here provided concerns the above research questions either commonly or individually and establishes a more refined view on the crucial aspects to be considered. While a detailed discussion of the research questions according to design science guidelines is presented here, a comprehensive methodological review of this work according to [57] is provided in the next section.

The current state of B2Bi can be characterized as follows:

- *No adoption of dedicated interoperability technologies that operate on low-cost Internet communication protocols.*
  The data provided by the Odette survey on *Data Exchange via Portals and EDI*[10] as well as the aims of the *auto-gration* project[12] clearly show that low-cost Internet communication protocols are not commonly used. This is particularly true for WSDL-based Web Services. Although this technology is conceptually a natural fit for integration scenarios that suffer from interoperability and heterogeneity problems, it is not even mentioned as an implementation option in the Odette analysis for *File Transfer Alternatives for B2B Data Exchanges*. This observation is also backed by the analysis of [1].

- *Integration is performed without using standardized process model formats for agreements.*
  The lack of standardized process model formats is one of the key observations of the study in [1] and is reflected by the fact that hardly any reputable B2Bi community such as RosettaNet, Odette, OAGi or papiNet provide scenarios or guidelines for using process model standards. When provided, B2Bi scenario descriptions offer ad-hoc visualizations that may or may not be based on well-known modeling languages and typically rely heavily on prose for clarifying semantics. As a consequence, these descriptions typically lack common meaning and any formal underpinning. This observation is also backed up by [68] that identify *standardization of modeling approaches* as a core issue for business process modeling in a survey among 64 business process modeling experts from academia, tool vendors and practitioners.

- *The generation of implementation artifacts from contractual process agreements is not sufficiently supported.*
  A model-driven process execution is identified as another core business process modeling issue in [68]. The lack of corresponding functions can be explained by the fact that B2Bi communities do not so far adopt process-based modeling at all. From an academic perspective, the generation of deployable artifacts for process execution is not a solved problem either. There are approaches that derive orchestration processes (in particular BPEL-based processes) from

high level process descriptions (cf. [152, 217]). However, such orchestration models still have to be completed with private integration logic manually before they can be deployed on execution engines. Considering the complexity of orchestration models, this is a challenging and error-prone task. The recently released BPMN 2.0 specification [150] is frequently said to provide everything needed to derive fully executable BPEL processes. However, there is so far no conclusive evidence that this is the case. The abstraction level of BPMN actually implies that further assumptions must be made for enabling the executability of process models, and choreographies in particular.

- *Integration architectures mix up business logic with control flow logic and thus foster lock-in to specific B2Bi vendor products.*
  As long as control flow logic and business logic are not separated from each other, integration partners will remain dependent on the integration functionalities of ERP and BSI providers for connecting message endpoints to business applications. Breaking this dependency cycle would necessitate the standardization of the relationship between cross-organizational message exchanges and business applications so that the control flow logic can be separated seamlessly from the business logic. This implies the use of process-based models as these provide the foundation for capturing the control flow. However, B2Bi communities still struggle with more basic communication technology problems or even do not recognize the need for such a separation. Rules for governing the control flow of message exchanges between BSIs and business applications is frequently perceived to be a private issue for the integration partners and is eventually left to B2Bi solution providers.

- *Current process specification formats do not provide suitable abstraction levels for cooperation between business users and software engineers.*
  Business process models created by domain experts focus on the actual business logic and not on technical soundness criteria such as executability or absence of deadlocks. Domain experts frequently do not even have the skills to think in these categories and use prose to document the semantics of process models in ambiguous ways. In contrast, deployment artifacts created by software engineers focus on technical aspects such as data type definitions, message endpoint bindings, or executable activity sequence specifications and thus fall short in making sense to non-technical personnel. As a result, issues such as *Business-IT-Divide*, *Compliance* and *Modeling Level of Detail* are identified in [68].

- *Limited visibility of the progress of B2Bi interactions.*
  In a poll performed by RosettaNet on the *High Tech and Electronics Industry* in early 2011[20], visibility throughout the supply chain is identified as the

[20]RosettaNet, *"High Tech and Electronics Industry"*, 2011, `http://www.rosettanet.org/dnn_rose/DocumentLibrary/tabid/2979/DMXModule/624/Command/Core_Download/Method/attachment/Default.aspx?EntryId=9851`, last access: 12/20/2011

most important integration challenge. However, visibility in terms of the progress of active interaction instances as well as the calculation of performance indexes implies the use of process models and requires the development of implementation techniques that allow for such information visibility. The adoption of process-based B2Bi systems is an enabler for this.

This thesis further strives to improve the current B2Bi landscape by achieving the following *goal state* of B2Bi:

- *The barrier for adopting process-based B2Bi is lowered.*
  The following factors contribute to eliminating process-based B2Bi adoption barriers. Firstly, the use of low-cost Internet technologies is enabled by identifying options for providing necessary B2Bi communication qualities such as security or reliability. As the B2Bi landscape is very diverse in terms of existing communication protocols, message gateways software and ERP systems, dedicated interoperability technologies, such as WSDL-based Web Services, are used for tackling interoperability and heterogeneity problems. Moreover, the use of stringent choreography and orchestration descriptions leveraging unambiguous execution semantics enables the reduction of misunderstandings between integration partners, faster implementation of B2Bi systems and improved compliance between implementation systems and B2Bi process models.

- *B2Bi choreographies can be used as standardized, technology-agnostic contractual obligations between integration partners.*
  The authors of [1] state that *"without [..stringent, vendor-independent business process standards], implementations can remain costly and incompatible, making a critical barrier to achieving the high level alignment and flexibility for agile supply chains"*. Similarly, *standardization* is identified in [68] as the most significant business process modeling issue by practitioners. Furthermore, the author of [124], who reports on a survey analyzing the impact of Information and Communication Technology (ICT) solutions on e-commerce with 5218 cases from ten industries in seven countries, states that *"standardized data exchange with the trading partners has a positive influence on labor productivity"*. However, standardization alone is not sufficient for the definition of suitable choreography models. In order to cater for the diverse set of BSI implementation options, choreography models must be implementation technology agnostic. Yet the execution semantics of the choreography models must uniquely define the set of valid message exchange sequences.

- *B2Bi is facilitated by an integration architecture that enables separation of the business logic from the control flow logic.*
  The business logic for creating, validating, and processing business documents as well as the functionality for triggering such events can be assumed to be available in existing business applications such as ERP systems. Isolating control flow logic requires the definition of standardized interfaces for consuming application

logic. Relying on such interfaces is a means to generate orchestration-based control flow implementations that are ready for deployment and do not have to be extended with business logic manually. Furthermore, an integration architecture that isolates control flow logic and defines the interactions with business applications provides a lever for better managing the control flow of interactions. Thus, the dependency on ERP or B2Bi vendor offerings for ensuring compliance between interaction implementations and choreography models can be reduced. In that sense, separation of business logic from control flow logic is an important means to reempowering the integration partners so that they regain control of their business processes.

- *B2Bi is facilitated by an integration architecture that enables tracking the progress of business interactions.*
  Timely and agile management of B2Bi interactions requires tracking the progress of business interactions. This includes applying process models that allow for the retrieval of already completed activities as well as the identification of the integration partner who is in charge of delivering the next message. In addition, there must also be some means for alerting partners about dangling processes.

- *Model-driven technologies can be used to derive orchestrations that govern the control flow of message exchanges.*
  Leveraging model-driven technologies for deriving implementation artifacts promises to provide the *"quick establishment of Information Systems (IS) integration"* as demanded by [1]. However, in order to be useful, orchestration models must be generated in a way that these are ready for immediate deployment. Otherwise, the effort for handcrafting automatically generated code just might be too high. In addition, leveraging model-driven technologies is not only advantageous in terms of time but also has the promise of cheaper implementation and improved conformity in orchestrations and choreography contracts. Even if automatically generated deployment artifacts do not fit in arbitrary Information Technology (IT) landscapes, these may still be very beneficial as reference implementations or testing systems.

- *B2Bi models can be stringently visualized without a significant amount of technical detail.*
  The business process modeling problems *Business-IT-Divide*, *Modeling Level of Detail* and *Ease of Use* identified in [68] all call for the visualization of process models. Whereas orchestration models consider implementation, choreography models look at the definition of implementation contracts that serve as a means of communication, not only between domain experts of the participating integration partners, but also between domain experts and software engineers. In light of this, the visualization of choreography models is a top priority. Some technical agreement detail needs to be hidden because such visual models should be amenable to domain experts. However, more technical choreography representations must be derivable where software engineers only have to fill in

technical detail without necessarily modifying the already agreed-upon message exchange sequences.

Having now defined the research question explicitly and through contrasting the current and goal state of B2Bi, the next section provides a review of the research method applied to answer this question.

1.3. Research Method

The target of this thesis is to identify a way to close the gap between business process models and private orchestrations of B2Bi systems. In considering the vast set of implementation options and user requirements for the type of B2Bi systems identified in section 1.1, enumerating all possible solutions to this problem and identifying the optimal solution is next to impossible. Following the *satisficing* principle of [200], it is the aim of this thesis to explore the use of choreography and orchestration technology. Yet [104] asserts that the development of *"IT solutions"* is supposed to follow *"design theories"*. The application of choreography and orchestration models as well as methods for deriving orchestrations from choreographies can be suitably supported by the *design-science* research method. The fundamental principle of design science research *"is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact"* [57].

Hevner et al. [57] recommend seven basic guidelines for performing high quality design-science research. In what follows, it will be outlined how this project corresponds to these principles.

Guideline 1: Design as an Artifact According to [57], *"design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation"* that is *"created to address an important organizational problem."*

The artifacts proposed in this thesis are models for specifying, executing and visualizing B2Bi choreographies as well as methods for analyzing and processing these.

A series of models is created in order to constrain and define the semantics of the ebBP B2Bi choreography language more precisely. As a result, ebBP can be used for the specification of unambiguous implementation contracts between B2Bi partners (cf. chapter 4). Firstly, the tasks of the communication roles of ebBP BusinessTransactions are captured as communicating state machines. Secondly, ebBP Regular (ebBP-Reg) and ebBP Shared State (ebBP-ST) are defined as binary ebBP choreography dialects that can be used as an implementation contract between exactly two partners. Thirdly, Sequential Multi-Party (SeqMP) choreographies and a corresponding framework are proposed for analyzing multi-party choreographies in order to extend the use of ebBP beyond implementation contracts. The concept of proposing more than one single language owes itself to conflicting requirements as identified in chapter 3.

The execution of binary choreographies is supported by means of an integration architecture that separates the control flow logic from the business logic (cf. chapter 5). Furthermore, a mapping from binary ebBP choreographies to BPEL is defined (also chapter 5).

Finally, guidelines for representing the different styles of choreographies using BPMN are proposed. In addition, the technical details that have to be filled in when deriving ebBP choreographies from visual models are identified.

Guideline 2: Problem Relevance *Problem relevance* is reformulated in [57] by stating that *"the objective of research in information systems is to acquire knowledge and understanding that enable the development and implementation of technology-based solutions to heretofore unsolved and important business problems."*

Beyond stressing once more that a lack of IT support endangers effective SCM or indeed the survival of the organization [51, 90], demonstrating problem relevance essentially means proving that a problem actually exists. This essentially corresponds to the identification of research questions as spelt out in the last section. The identified research problem is given further credibility by the fact that industry as well as academia are aiming to solve it. For example, RosettaNet has performed the so-called Multiple Messaging Services (MMS) and MCC efforts that resulted in guidelines for leveraging low-cost Internet technologies for message exchanges [171, 172] as well as guidelines for creating visual B2Bi models [173]. With regard to the MCC deliverables, core parts of their standards rely on the findings of this thesis. Moreover, a series of B2Bi standards, as analyzed in chapter 3, that offer some support for the identified research gap, are available. Furthermore, the second version of BPMN, released in January 2011, has added a dedicated choreography section. Finally, there is considerable academic research in the B2Bi field as discussed in chapter 7.

Guideline 3: Design Evaluation *"A design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve"* [57]. Thereby, *"the business environment establishes the requirements upon which the evaluation of the artifact is based"* [57].

In order to capture the requirements of the B2Bi domain, important B2Bi standards were included in an extensive literature review (cf. chapter 3) on B2Bi requirements. The development of such standards is typically driven by practitioners and software vendors and, therefore, can be assumed to contain commonly and frequently needed functionality. As a result, the requirements identified can be argued to represent a solid basis for shaping choreography models in order to fit business requirements.

Moreover, an analysis of 100 *RosettaNet Implementation Guides (RIGs)* of RosettaNet's RIG library has been performed. *"[A RIG] describes the specific business scenario(s), usage notes and lessons learned [when implementing RosettaNet standards]"* which is supposed to *"help reduce implementation time and accelerate adoption of the process scenario by sharing the experience of early implementers."*[21] Control

[21] `http://www.rosettanet.org/Support/ImplementingRosettaNetStandards/`

flow requirements of RIGs can be deduced from the business scenario descriptions and the overwhelming majority of RIGs is pretty simple in that regard. Only 44 out of 100 RIGs use hierarchical decomposition, 15 RIGs describe multi-party scenarios, 12 RIGs use loops, and 8 RIGs have parallel activities. Contrasting the B2Bi choreography models of this thesis with these RIG control flow requirements reveals that common B2Bi control flow requirements are covered in this thesis pretty well.

The choreography models have been further challenged by submitting these to the RosettaNet MCC team, which resulted in the acceptance of large parts of this dissertation project's results in two RosettaNet standards [172,173]. These standards cover the execution of ebBP BusinessTransactions using Web Services [172] on the one hand, and the representation of B2Bi choreographies using BPMN choreography notation [173] on the other. The conceptual foundation for the BPMN visualization is provided by the identified ebBP-Reg, ebBP-ST, and SeqMP choreography styles. Using BPMN, choreography models that follow these styles can be presented in a more user-friendly manner. Finally, the mapping of B2Bi choreographies to BPEL is evaluated by checking whether or not the created BPEL processes can be deployed and executed on the openESB[22] BPEL engine without the need for further modification.

Guideline 4: Research Contributions In [57], it is pointed out that *"effective design-science research must provide clear contributions in the areas of the design artifact, design construction knowledge (i.e., foundations), and/or design evaluation knowledge (i.e., methodologies)"*. In addition, [57] states that *"most often, the contribution of design-science research is the artifact itself."*

This holds particularly true for the artifacts described for guideline 1, which constitute the main contributions of this thesis. In what follows, these are listed. They can be categorized in the area of design construction and design evaluation knowledge:

- Firstly, this thesis develops a formal execution semantics for subsets of ebBP process models. To this point, no formal execution semantics has been available for ebBP.

- Secondly, the mapping from ebBP models to BPEL allows for almost arbitrary input graphs. In particular, irreducible loops are allowed for and hence the modeler is not forced to design ebBP models in a structured manner [76]. This is different from other research initiatives that simply ignore the problem, impose considerable limitations on the input models or leverage BPEL features that are barely available in today's engines. For example, [152] uses self-links to resolve irreducible loops which is not largely supported among current BPEL engines. Apart from allowing for almost arbitrary input models, the mapping provided in this work (also described in [190]) is new in combining a graph-like structure and threading-like process decomposition at the BPEL level. Such a

`RosettaNetImplementationGuides/tabid/2985/Default.aspx`, last access: 12/20/2011
[22]`http://openesb-dev.org/`, last access: 12/20/2011

strategy was not found in other research projects. In particular, the strategy is not captured in the overview of translation strategies between graph-oriented and block-oriented languages in [111].

- Thirdly, the visualization of B2Bi choreographies constitutes an evaluation of BPMN 2.0 choreographies with respect to its suitability for B2Bi modeling. The classes of ebBP-Reg, ebBP-ST and SeqMP were used to check BPMN's visualization capabilities by analyzing the availability of modeling constructs, the implementability of choreography grammar rules and by modeling several use cases. Some deficiencies such as missing role mapping features or missing expressions for capturing collaboration results have been detected, as well as the technical detail that has to be completed for deriving complete ebBP serializations of BPMN choreographies has been identified. It is worth noting that the usability of the visualization has not been an explicit goal of this thesis. However, the acceptance of the visualization proposal by the RosettaNet MCC team suggests that the visualization makes sense to practitioners and software vendors.

- Fourthly, an integration architecture for performing binary B2Bi choreographies that truly separates the control flow logic and the business logic is proposed. This separation also enables the generation of fully deployable BPEL processes. It is important to note here that this is not the same as conceptually identifying public and private processes for B2Bi interactions, which can be considered to be common knowledge in the B2Bi domain. It rather concerns the definition of interfaces as well as interaction rules for coupling public processes to private processes using orchestrations.

- Fifthly, the integration style proposed here combines asynchronous and synchronous coordination semantics in a new way, where messages are transmitted synchronously while the processing of messages is performed asynchronously. Exchanging messages synchronously simplifies protocol design as the sender of the message immediately knows the technical result of transmission. This concept is particularly useful as it significantly reduces the effort required for analyzing models using techniques such as model checking (see section 5.1). On the contrary, message processing that may require considerable time is performed asynchronously in order to prevent tight coupling. Based on these simple primitives, exchange protocols are defined that ensure consistent outcomes for ebBP BusinessTransactions as well as a synchronized view of integration partners on the progress of ebBP BusinessCollaborations. In order to achieve synchronized BusinessCollaboration views when advanced control flow features are used, messages that carry business semantics are distinguished from messages that only serve as coordination messages. If needed, such coordination messages are used to align the views of the integration partners, which allows for modeling scenarios that are disallowed in other approaches. However, it is vital to note that coordination messages do not affect the business outcome of interactions.

Guideline 5: Research Rigor *"Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact"* where *"design-science research often relies on mathematical formalism to describe the specified and constructed artifact"* [57].

Rigor has been applied to this work in several ways.

Firstly, established technologies, languages and formalisms have been used that provide a solid basis to begin research. Web Services have been designed as dedicated interface technology and, therefore, promise to ideally match B2Bi requirements. Furthermore, the models for representing choreographies and orchestrations have been based on international standards. Moreover, state machines have been used for formalizing the tasks of BusinessTransaction communication roles as well as underlying paradigm for ebBP-Reg, ebBP-ST, and SeqMP. State machines have a very long tradition in system formalization and give access to advanced analysis tools such as model checkers.

Secondly, ebBP-Reg, ebBP-ST and SeqMP have all formally been captured and practically modeled, which allows for reciprocal detection of design and formalization errors. Similarly, a formal execution semantics for binary ebBP models has been designed.

Thirdly, the mapping from ebBP models to BPEL has been validated by manually deriving orchestrations from ebBP use cases according to the defined mapping rules. Furthermore, subsets of the mapping rules have been implemented prototypically in order to rule out *manual bias* in the application of the mapping rules.

Fourthly, the analysis framework for SeqMP has prototypically been implemented and tested through real-world and artificial use cases.

Fifthly, the algorithm for deriving role projections for SeqMP choreographies has been formally analyzed.

Finally, challenging artifacts by submitting them to an experts forum, in this case, the RosettaNet community, is a further aspect of rigorous scientific investigation.

Guideline 6: Design as a Search Process This guideline is captured in [57] more precisely as *"the search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment"*.

This dissertation project started with the exploration of simple B2Bi scenarios and how these can be captured and implemented. From this it was clear that standardization plays a key role in the B2Bi domain and that international standards should be applied in as much as this is possible. An extensive literature review then revealed a large set of diverse requirements and it was obvious that not all requirements can be met by one single approach. Supporting communication between integration partners and streamlining implementation is of paramount importance for the selected type of B2Bi systems. As a consequence of this, the use of ebBP as an implementation contract as well as Web Services technology including WS-* and

BPEL was explored. The BPMN representation was finally created to demonstrate how a visual notation for B2Bi choreographies may look and to ensure amenability of this thesis' B2Bi choreography styles to realistic development projects.

Guideline 7: Communication of Research According to [57], *"design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences."*

The technology-oriented audience is served in two different ways. Firstly, formalizations of B2Bi choreographies as well as formal execution semantics for binary B2Bi choreographies are provided. Secondly, prototypic implementations of choreographies and of the multi-party analysis framework are described. Thirdly, the following list of technical reports as well as workshop, conference, and journal papers have been published that cover much of the technical detail for this thesis (in chronological order):

1. Andreas Schönberger, *Modelling and Validating Business Collaborations: A Case Study on RosettaNet*, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, Nr. 65, Technical Report University of Bamberg, 2006; [177]

2. Andreas Schönberger and Guido Wirtz, *Realising RosettaNet PIP Compositions as Web Service Orchestrations - A Case Study*, The 2006 International Conference on e-Learning, e-Business, Enterprise Information Systems, e-Government, & Outsourcing (CSREA EEE'06), Las Vegas, Nevada, USA, 2006, 141-147; [185]

3. Andreas Schönberger and Guido Wirtz, *Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions*, The 2006 International Conference on Software Engineering Research and Practice (SERP'06), Las Vegas, Nevada, USA, 2006, 329-335; [186]

4. Andreas Schönberger and Guido Wirtz, *Taxonomy on Consistency Requirements in the Business Process Integration Context*, Proceedings of 2008 Conf. on Software Engineering and Knowledge Engineering (SEKE'2008), Knowledge Systems Institute, 2008; [187]

5. Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger and Guido Wirtz, *QoS-Enabled B2B Integration*, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, Nr. 80, Technical Report, University of Bamberg, 2009; [16]

6. Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger and Guido Wirtz, *QoS-Enabled Business-to-Business Integration Using ebBP to WS-BPEL Translations*, Proceedings of the IEEE 2009 International Conference on Services Computing (SCC), Bangalore, India, 2009; [181]

7. Andreas Schönberger and Guido Wirtz, *Using Variable Communication Technologies for Realizing Business Collaborations*, Proceedings of the 5th 2009 World Congress on Services (SERVICES 2009 PART II), International Workshop on Services Computing for B2B (SC4B2B), Bangalore, India, 2009; [188]

8. Andreas Schönberger, Christian Wilms and Guido Wirtz, *A Requirements Analysis of Business-To-Business Integration*, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, Nr. 83, Technical Report, University of Bamberg, 2009; [184]

9. Christoph Pflügler, Andreas Schönberger and Guido Wirtz, *Introducing Partner Shared States into ebBP to WS-BPEL Translations*, Proc. iiWAS2009, 11th International Conference on Information Integration and Web-based Applications & Services, 14.-16. December 2009, Kuala Lumpur, Malaysia, 2009; [160]

10. Andreas Schönberger, *The CHORCH B2Bi Approach: Performing ebBP Choreographies as Distributed BPEL Orchestrations*, Proceedings of the 6th World Congress on Services 2010 (SERVICES 2010), Second International Workshop on Services Computing for B2B (SC4B2B), Miami, Florida, USA, 2010; [178]

11. Andreas Schönberger, Guido Wirtz, Christian Huemer and Marco Zapletal, *A Composable, QoS-aware and Web Services-based Execution Model for ebXML BPSS BusinessTransactions*, Proceedings of the 6th 2010 World Congress on Services (SERVICES2010), Fourth International Workshop on Web Services and Cloud Services Testing (WS-CS-Testing 2010), Miami, Florida, USA, 2010; [192]

12. Johannes Schwalb, Andreas Schönberger, *Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations*, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, Nr. 87, Technical Report, University of Bamberg, 2010; [196]

13. Andreas Schönberger, Christoph Pflügler and Guido Wirtz, *Translating Shared State Based ebXML BPSS models to WS-BPEL*, International Journal of Business Intelligence and Data Mining - Special Issue: 11th International Conference on Information Integration and Web-Based Applications and Services in December 2009, 2010, 5, 398 - 442; [182]

14. Andreas Schönberger and Guido Wirtz, *Towards Executing ebBP-Reg B2Bi Choreographies*, Proceedings of the 12th IEEE Conference on Commerce and Enterprise Computing (CEC'10), Shanghai, China, 2010; [190]

15. Johannes Schwalb, Andreas Schönberger and Guido Wirtz, *Approaching Interoperability Testing of QoS based on WS-* Standards Implementations*, The 4th Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'10), co-located with 8th IEEE European Conference on Web Services (ECOWS 2010), Ayia Napa, Cyprus, 2010; [197]

16. Andreas Schönberger and Guido Wirtz, *Sequential Composition of Multi-Party Choreographies*, Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA'10), Perth, Australia, 2010; [189]

17. Andreas Schönberger, *Do We Need a Refined Choreography Notion?*, Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS), Karlsruhe, Germany, February 21-22, 2011, CEUR-WS.org, 2011, 16-23; [179]

18. Jörg Lenhard, Andreas Schönberger and Guido Wirtz, *Streamlining Pattern Support Assessment for Service Composition Languages*, Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS), Karlsruhe, Germany, February 21-22, 2011, CEUR-WS.org, 2011, 112-119; [94]

19. Matthias Geiger, Andreas Schönberger and Guido Wirtz, *A Proposal for Checking the Conformance of ebBP-ST Choreographies and WS-BPEL Orchestrations*, Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS), Karlsruhe, Germany, February 21-22, 2011, CEUR-WS.org, 2011, 24-25; [45]

20. Matthias Geiger, Andreas Schönberger, and Guido Wirtz, *Towards Automated Conformance Checking of ebBP-ST Choreographies and Corresponding WS-BPEL Based Orchestrations*, Proceedings of 2011 Conf. on Software Engineering and Knowledge Engineering (SEKE'2011), Miami, Florida, USA, Knowledge Systems Institute, 2011; [46]

21. Simon Harrer, Andreas Schönberger and Guido Wirtz, *A Model-Driven Approach for Monitoring ebBP BusinessTransactions*, Proceedings of the 7th World Congress on Services 2011(SERVICES2011), Washington, D.C., USA, 2011; [54]

22. Andreas Schönberger, Johannes Schwalb and Guido Wirtz, *Has WS-I's Work Resulted in WS-* Interoperability?*, Proceedings of the 9th 2011 International Conference on Web Services (ICWS 2011), Washington, D.C., USA, 2011; [183]

23. Jörg Lenhard, Andreas Schönberger and Guido Wirtz, *Edit Distance-based Pattern Support Assessment of Orchestration Languages*, Proceedings of On the Move 2011 Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, Oct 19-21, 2011, Springer, 2011; [93]

24. Andreas Schönberger, *Visualizing B2Bi Choreographies*, Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA'11), Irvine, California, USA, December 12-14 2011; [180]

25. Andreas Schönberger and Guido Wirtz, *Configurable Analysis of Sequential Multi-Party Choreographies*, forthcoming in International Journal of Computer Systems Science and Engineering, 2011; [191]

The perspective of the management-oriented audience is taken into consideration in several ways. The benefits of this thesis are shown through the research question outlined previously as well as the requirements to be fulfilled in chapter 3. Furthermore, the ebBP modeling styles are constructed in such a way that users can create almost arbitrary graphs without threatening the derivation of BPEL-based implementations. It is only for parallel control flow structures, which are pretty uncommon in B2Bi scenarios, where a structured modeling approach is required. Limiting the number of constraints on input models lowers the barrier for users to adopt a modeling or specification language. Furthermore, concepts that are well-known from the B2Bi domain such as RosettaNet PIPs or ebBP BusinessTransactions are reused. In addition, the visual BPMN representation of ebBP models provides a more user-friendly means of specifying B2Bi interactions. Thereby, the formalizations of the respective ebBP modeling styles are translated into examples on how to connect visual modeling constructs. Moreover, a series of samples with gradually rising complexity is provided so as to smooth the introduction of the BPMN representation. Moreover, this thesis was partly presented during participation in the RosettaNet MCC effort where the author of this work was leading the Web Services sub-team of MCC phase 1 and was actively contributing content during MCC phase 2, which eventually resulted in the release of two international RosettaNet standards:

1. RosettaNet, *Message Control and Choreography (MCC) - Profile-Web Services (WS), Release 11.00.00A*, 2010; [172]

2. RosettaNet, *RosettaNet Methodology for Creating Choreographies, R11.00.00A*, 2011; [173]

Finally, managerial overview presentations of this dissertation project's results were given at 112th EDIFICE industry conference in Amsterdam in November 2010 and to the broader RosettaNet community through Web casts.

1.4. Outline

The rest of this thesis is structured as follows: Chapter 2 gives an overview of relevant technologies. Chapter 3 describes the results of an extensive literature review on B2Bi requirements. These were first collected for arbitrary B2Bi systems and then tailored to the selected type of B2Bi systems as previously outlined. Chapter 4 starts out with a discussion of ebBP as a choreography representation format. Thereafter, different strategies for bridging the gap between business process models and private orchestrations are discussed where choreographies are either used as loose and imprecise identification of document exchange scenarios (which will be denoted as *cartography choreographies*) or as unambiguous and complete specifications of admissible message exchange sequences (which will be denoted as *strict choreographies*). The basic integration architecture is then introduced, which includes important assumptions about the execution environment. Following this, an execution model for ebBP BusinessTransactions (BTs) as an atomic building block of B2Bi choreographies is outlined. ebBP-Reg, ebBP-ST as well as SeqMP are then introduced as separate strict choreography modeling styles that accommodate different combinations of the requirements identified in chapter 3. For all modeling styles, a formal definition is given. For the binary modeling styles ebBP-Reg and ebBP-ST, an execution semantics for streamlining implementation is given, while an analysis framework for the multi-party modeling style SeqMP is provided. This is due to the fact that B2Bi implementations are typically performed on a bilateral basis whereas SCM suggests considering the effects of message exchanges on the overall partner network. In chapter 5, the main software artifacts for realizing the integration architecture are described as well as the implementation of the execution model for ebBP BusinessTransactions. This is followed by a description of the implementation of the execution semantics defined for B2Bi choreographies (ebBP BusinessCollaborations) that are composed from BusinessTransactions. Chapter 6 presents the BPMN visualization of ebBP-Reg, ebBP-ST and SeqMP in a user-friendly way and discusses compliance with the BPMN standard as well as deficiencies detected. Chapter 7 extensively discusses related work structured along the topics presented in chapters 3, 4, 5 and 6. Finally, chapter 8 concludes by discussing how the identified research problem could be addressed and points out directions for future work.

The appendices A, B, C and D complement the thesis with content that does not fit into the main part because of the format or because the author of this thesis is not the original author of the text. The author takes full responsibility for the contents nonetheless. The relationship of the appendices with the main part is highlighted in the respective chapters of the main part.

2. Technological Background

This chapter provides an overview of the most important technologies used in this work, i.e., Web Services as communication technology (section 2.1), choreographies for specifying publicly visible message exchanges between integration partners (section 2.3), and orchestrations as implementation technology (section 2.2). In addition to BPMN, ebBP and BPEL, alternative choreography and orchestration languages as well as the ebXML B2Bi framework (section 2.4) are briefly described. The justification for choosing BPMN, ebBP and BPEL instead of these alternative technologies is provided in the next chapter and throughout the chapters that cover the respective abstraction layers.

2.1. Web Services and WS-*

It has become popular to denote almost any kind of activity as a *service*. In case some computing is associated with the respective activity, the term *"e-service"* is preferred. More accurate definitions associate the notion of *value* with an e-service: *"If you can imagine a way of electronically delivering something of value to a customer that will solve some problem or provide some usefulness to make their life easier, you have a viable example of an e-service"* [203]. Such business-oriented characterizations are helpful as an introduction to the field of electronic services. Unluckily, they are frequently confused with *"Web Services"* in a technical sense and any kind of *value-generating* service accessible via the Internet is denoted as a Web service. However, this does not comply with the original definition of the term *"Web Services"* as special kind of technology (given by the World Wide Web Consortium (W3C)):

> *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."* [221]

The most important parts of this definition for the purpose of this work are the machine-processable interface description and the prescribed interaction that complies with this description. These two parts bear the declarative power of Web Services technology offered to application developers who just provide a description of a service interface and the actual generation and exchange of messages between service provider and service consumer then is done by some Web Services frameworks. This work adopts the definition of the W3C. The term *"Web service"*, used for referring to a particular service, will always imply the use of WSDL (see below) for service description, the use of SOAP (see below) as message container format, the use of Extensible Markup Language (XML) for serializing contents, and the use of some Internet protocol like Hypertext Transfer Protocol (HTTP) as transport medium. The term *"Web Services"* will be used accordingly to refer to the technology itself, i.e., using WSDL, SOAP, XML and an Internet protocol for implementing a Web service.

While XML and Internet protocols like HTTP or FTP are assumed to be familiar to the reader, SOAP and WSDL are briefly described.

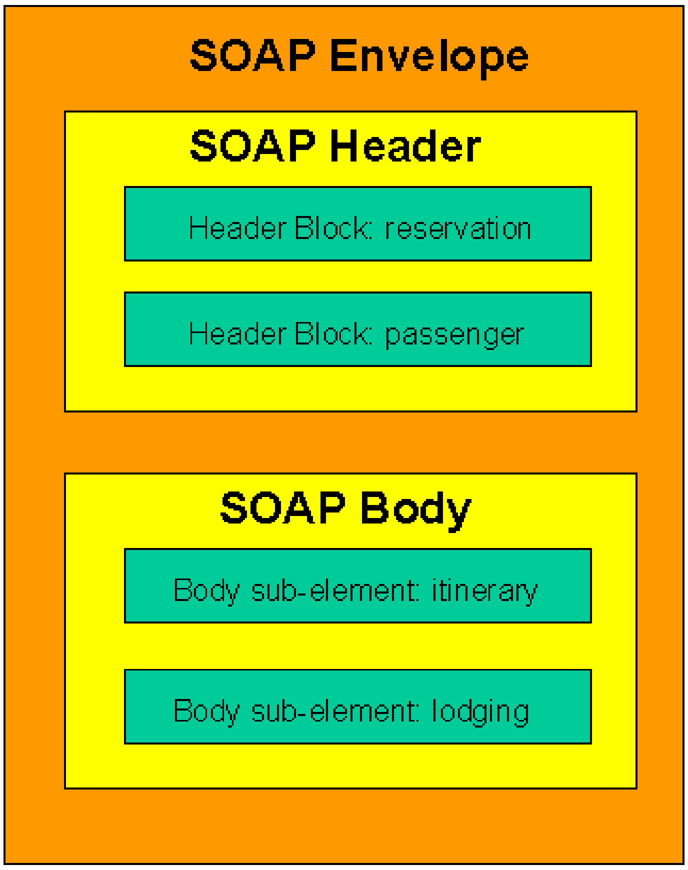

SOAP Envelope
SOAP Header
Header Block: reservation
Header Block: passenger
SOAP Body
Body sub-element: itinerary
Body sub-element: lodging

Figure 2.1.: SOAP Message Structure, taken from [225]

SOAP [225] is a messaging format for conveying XML content specified by W3C. The basic idea of SOAP is to define a message container that can be bound to a variety of communication technologies such as HTTP, SMTP or even Java Message Service (JMS) and still offers the possibility to introduce protocol aspects in the message container's header. Like that, transport features of existing communication technologies can be used for implicitly defining protocol aspects of message exchanges while application level messaging requirements still can be encoded into the SOAP message header in case the existing features are not sufficient. Moreover, the separation of the message container from communication technology enables using multiple communication technologies during the exchange of one single message. The benefit of SOAP therefore is not the definition of a full-featured communication protocol, but rather the provision of a standardized format for reconciling application level protocol requirements with existing communication protocols.

Figure 2.1 shows the basic structure of a SOAP message container that consists of a `header` and a `body`. While the `header` is optional, inclusion of a `body` element is compulsory. SOAP allows for carrying the `body` across multiple SOAP nodes from a SOAP sender, via optional SOAP intermediaries to an ultimate SOAP receiver. The `body` carries the actual payload and is assumed to be the same for one such end-to-end exchange. Conversely, the `header` data may change from hop to hop. Moreover, it is possible to use different communication protocols for each hop. Assume that a message is supposed to travel from some sender A via some intermediary B to an ultimate receiver C. Then, HTTP could be used for the exchange between A and B who processes and alters the header and then uses SMTP to relay the message to C. Note that not only the contents of the `body` element are optional, but the contents of the `header` as well. So, depending on the application needs, some header elements may require storing a message persistently while some other header elements may require adding the processing time for each message to the header. Only some generic attributes are defined for header elements such as `mustUnderstand` that requires any SOAP node to emit an error in case the contents of a header element are not understood. In consequence, adopters of SOAP have both the flexibility and the burden to define how SOAP messages are processed, in other words, to define a communication protocol using the SOAP format. However, defining a communication protocol is far from trivial so that the average application developer is not the target user group of SOAP. Instead, providers of advanced messaging solutions such as standardization organizations or solution providers are the typical users that have the resources to really leverage the flexibility of SOAP. Examples of how SOAP can be leveraged to define higher-level messaging functionality are basic Web Services technology itself, the various WS-* standards introduced below or ebXML Messaging Services (ebMS) [129, 136].

The WSDL language has been proposed by the W3C as description format for Web Services. Although version 2 of the WSDL standard [226] already has been released in 2007, the most important WS stack implementations such as Metro[1] or Windows Communication Foundation[2] still stick to version 1.1 [219]. More importantly, other specifications that build upon WSDL also adopt version 1.1 such as BPEL [137] or JAX-WS[3]. Consistently, this work relies on WSDL 1.1, too, and therefore the term *"WSDL"* always will refer to version 1.1 throughout the rest of this work. The WSDL language consists of six core elements: `types`, `messages`, `portTypes`, `bindings`, `services` and `ports`. `Types` define the different types of

[1]`http://metro.java.net/`, last access: 12/20/2011
[2]`http://msdn.microsoft.com/en-us/netframework/aa663324`, last access: 12/20/2011
[3]`http://jcp.org/en/jsr/summary?id=jax-ws`, last access: 12/20/2011

XML content that can be exchanged and the common format to describe those types is XML Schema Definition (XSD) [222]. Although the WSDL standard theoretically allows to adopt other type systems using extensibility mechanisms, today's WS stacks mainly rely on XSD definitions. WSDL `messages` consist of multiple message `parts` where each `part` is assigned a particular XML `type`. `Messages` are the basic primitive to define what kind of content can be exchanged between service provider and consumer. WSDL `operations` then can be used to define the direction and cardinality of message exchanges and optional fault messages. While `one-way operations` define the possibility to send a single message to the service provider, `request-response operations` define the exchange of a single request message to the provider and a response message to the consumer [219, section 2.4]. Note that indeed only one `message` may be exchanged per direction. If multiple parameters are required for an *operation call* then multiple `message parts` are defined or these are encoded as sub-elements of an XML type. WSDL also defines the `solicit-response` and `notification` types of `operations` as symmetric counterparts to `request-response` and `one-way`, but these are discouraged for practical use by practitioner communities (cf. [237]). A `portType` consists of several `operations` and captures the *functionality* of a service in the sense of message exchange capabilities. Taken together, the `types`, `messages` and `portType` of a WSDL file can be referred to as the *interface definition* part of a Web service. This part is abstract because neither the transport protocol for exchanging messages nor the endpoint of a service are known. This missing information is filled in by the WSDL `binding`, `service` and `port` elements. A `binding` defines how the `messages` and `operations` of a particular `portType` are to be transported. The most common binding for a Web service is SOAP via HTTP, but FTP, SMTP or other protocols are eligible as well (although barely supported in practice). A `port` definition then defines one endpoint address for a particular service, typically as a Uniform Resource Locator (URL), and the `binding` to be used (which indirectly also defines the portType). Finally, a WSDL `service` definition is used to specify one or more ports of the same service.

As a dedicated interface technology, Web Services bear the potential to bridge different platforms and programming languages, i.e., the types of messages and admissible interactions between service consumer and service provider are specified, but not the implementation of neither the consumer nor the provider. Consider the sample Web Services interaction between a Microsoft .NET service client (or consumer) and a Java service provider (or service) as depicted in figure 2.2. The service provider's platform, say, a GlassFish application server, provides an Enterprise JavaBeans (EJB) container as runtime environment for the service implementation and publishes a WSDL file that corresponds to the message exchanges offered by the implementation. In addition, an endpoint for consuming incoming messages is installed using the Web Services stack (WS stack) Metro which is integrated into the GlassFish server. Upon receipt of a SOAP message, the WS stack deserializes the XML contents into a Java representation and relays the message to the service implementation. If applicable, the results of message processing are collected, serialized into XML representation and then sent back as a SOAP message to the service consumer. The .NET client works

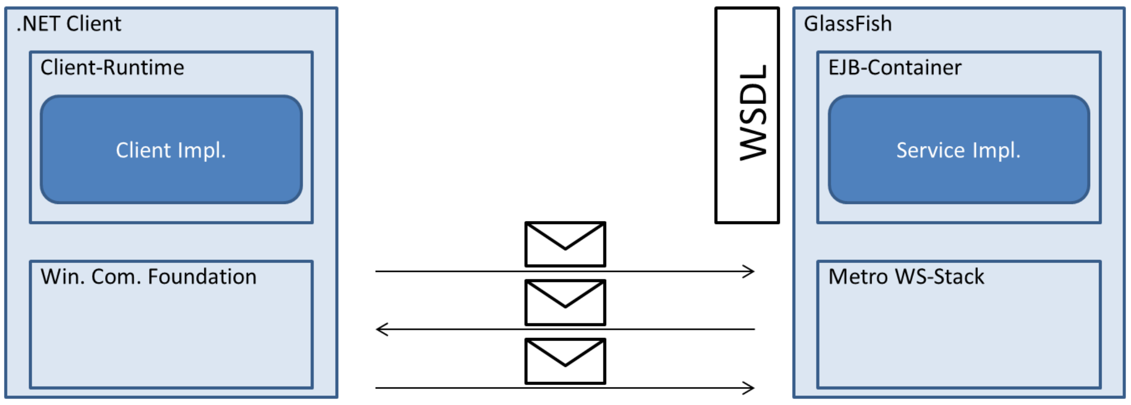


Figure 2.2.: Sample Web Services Interaction between Heterogeneous Platforms

symmetrically and also leverages a WS stack implementation to serialize/deserialize platform specific message representations into/from XML and to perform the SOAP communication on behalf of the client implementation. In order to produce messages that are understood by the Java service, the .NET client uses the service's WSDL file (which may have been exchanged beforehand). The power of Web Services technology in this scenario is leveraging standard tooling of solution providers to generate implementation skeletons for consumers and providers from WSDL descriptions as well as having standard proxies implement the cross-platform message exchanges using SOAP. In an ideal world, an application developer thus never is forced to process a SOAP message manually. Therefore, pure SOAP messaging technologies like ebMS are not true Web Services technologies because no machine-processable interface description is available that could be leveraged for auto-generating message exchanges.

Obviously, the full potential of Web Services unfolds only if reasonable interoperability between different vendors' implementations of Web Services standards is provided [122]. This is not as easy as it may seem to be at first sight because Web Services technology continuously grew in complexity. While early Web Services standards were supposed to provide lightweight and stateless interactions between systems, the wish to adopt interoperable communication for advanced application areas like Enterprise Application Integration (EAI) or B2Bi quickly resulted in the development of additional Web Services standards beyond WSDL and SOAP. Figure 2.3 shows the most important layers and standards that current WS stacks are supposed to support where each box represents a standard or functionality.

At the lowest level (bottom of the figure), support for binding SOAP exchanges to various existing (Internet) communication protocols like HTTP or SMTP is expected. Note that these are not actual Web Services standards, but they are included as crucial part of Web Services communication. Messages that are exchanged at this level, e.g., an HTTP request, will be referred to as *"transport level"* messages in the rest of this work.

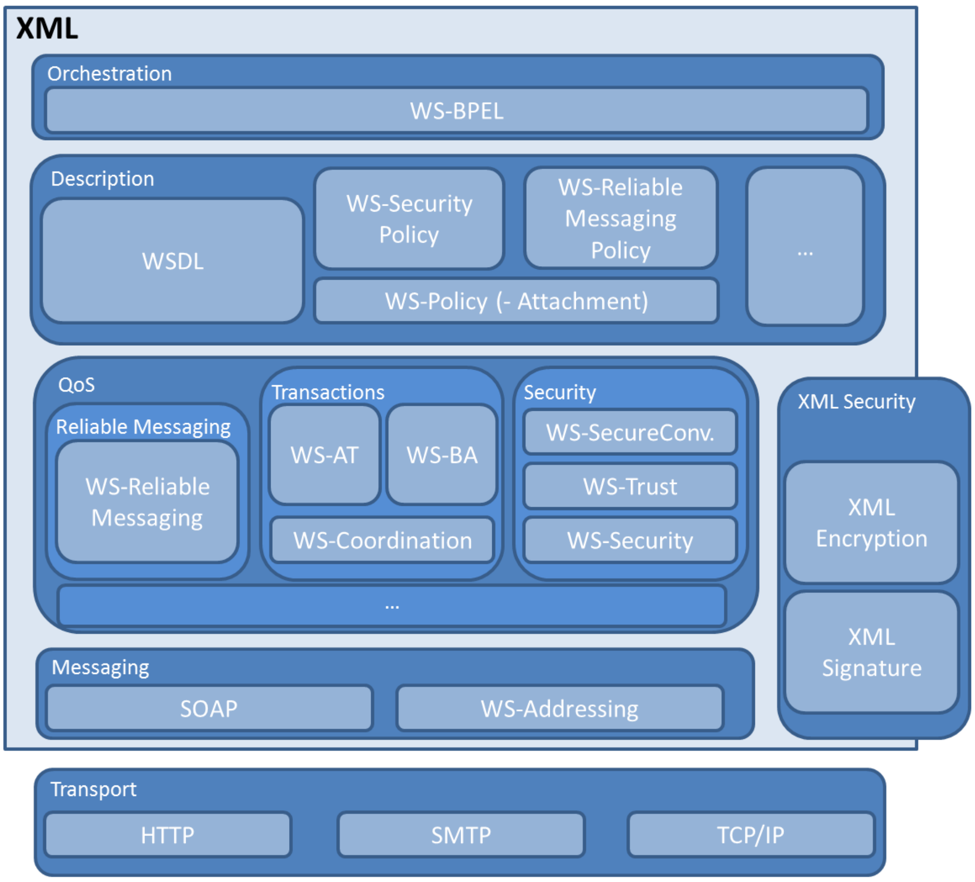


Figure 2.3.: Sample Web Services Stack, adapted from [196]

Contrary to these *transport* protocols, all Web Services standards heavily rely on the use of XML which is represented by the big rectangle labeled *'XML'* that contains the majority of standards.

The lowest-level layer of Web Services standards (lower part of the *'XML'* rectangle) is denoted *'Messaging'* and contains SOAP as the core format for exchanging Web Services messages. WS-Addressing [224] is a specialized format for referencing Web Services endpoints, interaction instances and corresponding SOAP messages. Typically, the format is used to define SOAP message headers for a particular application scenario, e.g., the implementation of a reliability protocol. Similar to SOAP, WS-Addressing does not define a protocol itself, but just a format for doing so. Messages that are exchanged at this level, i.e., SOAP messages, will be referred to as *"messaging level"* messages in the rest of this work.

The *'Quality of Service (QoS)'* layer on top of the *'Messaging'* layer contains a series of standards that implement advanced communication qualities such as reliable messaging (left box), transactions (center box), security (right box) and others (simply denoted as ellipsis character). A common characteristic of these standards is that they define processing instructions for SOAP messages in terms of defining the type, sequence and/or transformations of SOAP header elements, SOAP body contents and SOAP messages. For example, the WS-Reliable Messaging standard [143] defines how to run a reliability protocol between a service consumer and service provider (over possibly multiple SOAP intermediaries). Figure 2.4 shows a sample run of the WS-ReliableMessaging protocol (each arrow represents a SOAP message) that consists of the establishment of the reliable messaging sequence (first two messages), several payload and acknowledgment messages (next six messages) and the termination of the sequence (last two messages). Note that applying WS-ReliableMessaging implies exchanging at least six SOAP messages per Web Services call, i.e., even if a service consumer just sends a single application level message to the service provider several SOAP messages are used for ensuring delivery (or detecting delivery errors if not possible). Similarly, WS-Coordination [141], WS-AtomicTransaction [139]

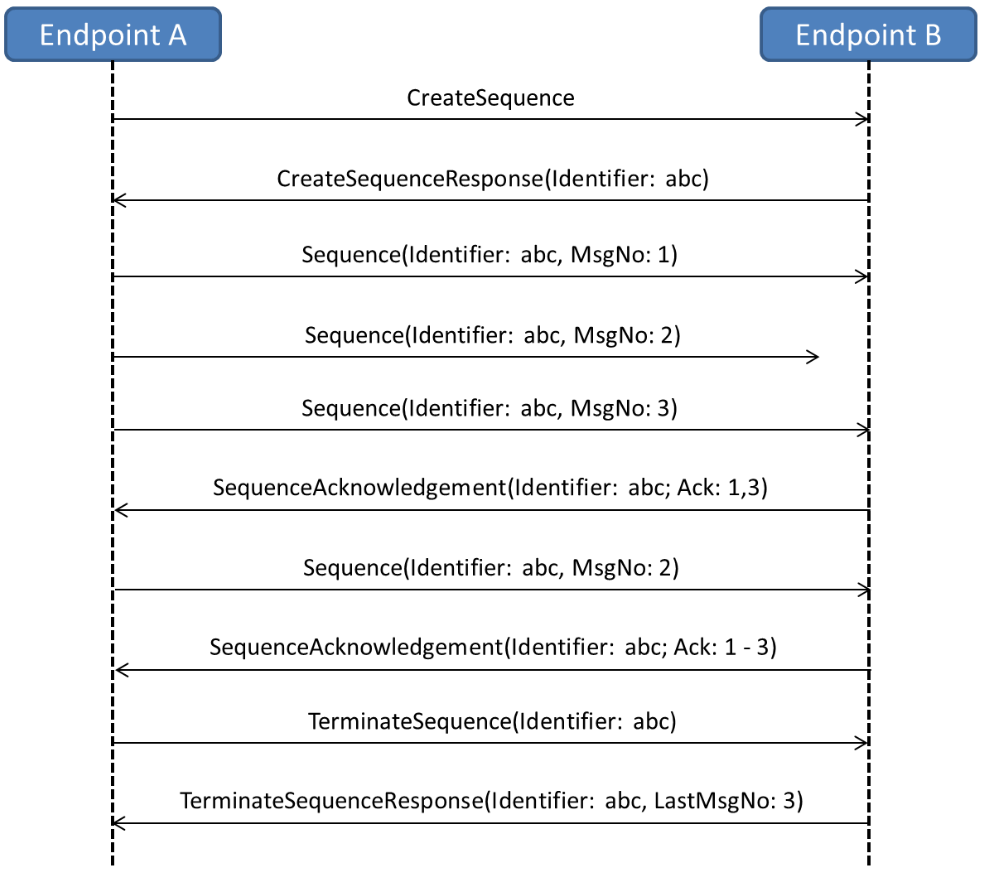

Endpoint A
Endpoint B
CreateSequence
CreateSequenceResponse(Identifier: abc)
Sequence(Identifier: abc, MsgNo: 1)
Sequence(Identifier: abc, MsgNo: 2)
Sequence(Identifier: abc, MsgNo: 3)
SequenceAcknowledgement(Identifier: abc; Ack: 1,3)
Sequence(Identifier: abc, MsgNo: 2)
SequenceAcknowledgement(Identifier: abc; Ack: 1 - 3)
TerminateSequence(Identifier: abc)
TerminateSequenceResponse(Identifier: abc, LastMsgNo: 3)

Figure 2.4.: Sample WS-ReliableMessaging Protocol Run, taken from [143, 196]

and WS-BusinessActivity [140] commonly provide a framework for implementing short-term and long-running transaction support for Web Services interactions.

The *'Security'* box shows the most important Web Services standards in terms of securing message exchanges. WS-Security [135] defines how to apply existing *'XML Security'* standards, (box on the right edge), i.e., XML Encryption [220] and XML Signature [228], to SOAP messages. WS-Trust [146] specifies the exchange of security tokens across different trust domains and thus enables secure interactions without exchanging digital security certificates beforehand manually. WS-SecureConversation [144] finally builds upon WS-Trust for describing how security contexts can be established that then serve for deriving session keys.

Note that all these different QoS standards are orthogonal to a large extent and XML namespace technology can be used to combine the formats of the respective standards within SOAP messages. Although there are some references among the standards, whether or not, say, WS-ReliableMessaging is combined with WS-Security is the application developer's choice. While this concept fosters flexibility by allowing (almost) arbitrary combinations of existing and ever emerging standards, it also causes complexity by leaving the question of correct integration of standards open. Consider the combination of WS-ReliableMessaging and WS-Security for providing secure and reliable message exchanges. Simply implementing security on top of a reliable channel is insufficient because security goals imply the existence of an attacker who may try to break security goals by tempering with the underlying reliability channel. Hence, a careful analysis of communication requirements and intricate combination of WS-ReliableMessaging and WS-Security is necessary for providing *secure reliable* communication which is far from trivial [4, 44].

The complexity of the various QoS standards calls for a convenient way for application developers to assert the realization of QoS properties for their interactions. This need is handled at the *'Description'* layer on top of the *'QoS'* layer of figure 2.3. The basic interaction of Web Services is covered by WSDL as described above and the WSDL `messages` defined as input or output of a WSDL `operation` are referred to as *"application level"* messages throughout the rest of this work. For advanced communication requirements, the WS-Policy [227] framework can be used to assert the necessary communication features. WS-Policy defines a format for prescribing the rules of interaction with a service (on top of WSDL definitions) as a so-called services policy. A services policy consists of one or mutually excluding policy alternatives that are composed of one or more policy assertions in turn. A policy assertion requires the application of a particular communication feature to the SOAP message exchanges that implement a particular Web Services interaction. For example, the so-called `RMAssertion` can be used to require the application of the WS-ReliableMessaging protocol. The association of these assertions with service `operations` is typically defined by including policy definitions within a particular WSDL file and referencing the intended policy within the service `binding` using `PolicyReferences`. Note that WS-Policy does not offer any specific assertions. Instead, a series of additional standards defines the assertions for dedicated purposes. For example, the `Web Services Reliable Messaging Policy Assertion` standard [142] defines assertions for requiring the use of the WS-ReliableMessaging protocol, among others the `RMAssertion` mentioned above. Similarly, WS-SecurityPolicy [145] offers a set of assertions for requiring the application of WS-Security, WS-Trust and WS-SecureConversation features. Note further that it is the task of the respective WS stack in use to process WS-Policy assertions included in the WSDL description of a Web service and to automatically generate and format the relevant SOAP messages. Again, the application developer just declaratively specifies the cross-platform message exchange features and never is encountered with a SOAP message (in an ideal world). Requiring application developers to manually add the features of QoS standards like WS-ReliableMessaging or WS-Security to SOAP messages is not true Web Services technology.

This observation also is the basis for pinning down the notion of *"WS-*"* that frequently is just used to denote all Web Services standards that start with the string "WS-".

> **WS-* standard**: For the purpose of this work, the term WS-* standard is defined to be a Web Services extension that implements a QoS feature for Web Services interactions by defining detailed SOAP message processing instructions. Moreover, a set of WS-Policy expressions for asserting the respective features is assumed to be available for a WS-* standard.

The last layer included in figure 2.3 is the orchestration layer with BPEL [137] as the most prominent representative of orchestration languages. The orchestration layer is used to define the sequences of application level message exchanges that make up a process-based application. Note that this layer is not core functionality of typical WS stack implementations and therefore makes up the *upper boundary* of the stack description. The characteristics of orchestration technology will be discussed in the next section.

Finally note that service discovery standards deliberately are not included in the description of WS stack functionality. In the early days of Web Services, Universal Description, Discovery and Integration (UDDI) [133] as means for implementing service registries was frequently proposed to be used by the service consumer to look up the description of a service from the registry and dynamically bind to the service. This kind of interaction scenario is of minor importance for the type of B2Bi system targeted in this work (cf. section 1.1) because integration partners are assumed to interact in a more or less stable relationship where at least the type of service interface to be used is known in advance (concrete endpoints still may be looked up via a service registry). Therefore, service registries and dynamic binding will not further be discussed in this work.

The features of a WS stack can be used to implement both synchronous and asynchronous coordination styles. Thereby, the behavior of the actual application client must be separated from the messaging proxy's behavior. For characterizing the application client's behavior, the terms *"synchronous interaction"* and *"asynchronous interaction"* will be used. For characterizing the messaging proxy's behavior, the terms *"synchronous communication"* and *"asynchronous communication"* will be used. When sending a message, an application client basically can either wait until an application response is returned by the receiver or can choose to perform some other activities in between. In the former case, the interaction is synchronous and asynchronous in the latter. This is to be separated from the communication semantics that a messaging proxy offers. If the messaging proxy forces the application client to block until the transmission of the message is accepted by the receiver then the communication is synchronous. If the messaging proxy just takes over the application message and transmits it (possibly) at a later point in time then the communication is asynchronous. Asynchronous communication hence implies that the sender does not exactly know when the application level message will be delivered. A typical synchronous communication technology is HTTP as the sending messaging proxy (typically) awaits an HTTP response code (for example *"200"* or *"202"* for successful transmission) that is communicated to the application client. A typical asynchronous communication technology is SMTP where an application client submits its message and does not know when (if ever) the receiver will get the message. The common binding for Web Services, SOAP via HTTP, also follows the synchronous communication paradigm (cf. SOAP 1.1[4], SOAP 1.2[5]). In practice, a Web Services interaction typically is performed without SOAP intermediaries so that synchronous communication also is available for a concrete Web service consumer or provider. The choice of communication semantics fundamentally influences the design of application level interactions. Knowing whether or not a message was transmitted successfully is the basis for persisting transactions and controlling local systems. If this kind of information is not provided by the messaging system and alignment with respect to message transmission success is needed then complex reliability or distributed commit protocols need to be performed at the application level which typically is not acceptable.

Note that synchronous/asynchronous interaction and synchronous/asynchronous communication can be combined almost orthogonally. For example, some application client A can send some message X to service B using synchronous communication and not expect any immediate application response. Hence, the benefit in using a synchronous channel only is that information about transmission success is available immediately. Then, B processes the message and sends back some response message Y at a later point in time using a new synchronous transmission. In the meantime, A may or may not perform other work. So, A and B interact asynchronously, but they use synchronous communication. Conversely, A could use an asynchronous transmission for sending X and not accept any other message until the response Y is returned using another asynchronous transmission. Then the interaction would be synchronous and the communication asynchronous.

Supporting interoperability throughout all WS stack layers is a strict requirement for realizing the Web Services promise of bridging communication across different platforms. In practice, this is not always true. Simple facts such as missing support for WSDL version 2.0 or the `solicit-response` and `notification` operation types of WSDL 1.1 by major WS stacks imply that even basic definitions may suffer from

[4]`http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383526`, last access: 12/20/2011
[5]`http://www.w3.org/TR/soap12-part2/#soapinhttp`, last access: 12/20/2011

interoperability issues. The Web Services Interoperability Organization (WS-I)[6] is dedicated to provide clarifications on such issues and to foster interoperability of Web Services by creating so-called *profiles* which provide rules for creating and processing WSDL files and SOAP messages. A WS-I profile documents *"[..]clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability"* [240]. The available profiles are the *Basic Profile* [238], the *Basic Security Profile* [239] and the *Reliable Secure Profile* [240]. Some of the profile contents refer to general aspects of Web Services communication, e.g., the Basic Profile forbids the use of the `solicit-response` and `notification operation` types and prescribes the use of WSDL 1.1. However, the majority of requirements defines constraints on the level of SOAP messages, i.e., the existence, order and content of XML elements within SOAP messages or the actual exchange of SOAP messages is described. For example, requirement R3203 of the Basic Security Profile defines that *"A TIMESTAMP [element in a SOAP message] MUST contain exactly one CREATED [XML element]"*. In addition to the profiles, the WS-I provides testing tools that take SOAP messages as input and check them for compliance to the profile requirements, but the profiles are authoritative. A core problem of the WS-I profiles in terms of security and reliability features is that that relation between WS-Policy assertions and the corresponding SOAP messages is not provided. In its section 5.1.1, the Basic Security Profile explicitly allows for out of band agreement for specifying the use of WS-Security features. Moreover, it states in several sections (9, 10, 13.1) that *"[..]no security policy description language or negotiation mechanism is in scope for the Basic Security Profile[..]"*. The Reliable Secure Profile recommends (though not requires) the use of WS-ReliableMessaging Policy for configuring the use of WS-ReliableMessaging in its section 2.4, but it does not define the relation between policy assertions and SOAP messages either. So, while WS-I for sure has contributed significantly to Web Services interoperability, its deliverables are not sufficient to guarantee interoperability in practice.

Corresponding systematic analyses [183,196] created during this work's dissertation project reveal that interoperability issues for security and reliability features indeed are serious.

To summarize, the most important benefits and drawbacks of using Web Services technology are given. The most important benefits are:

- Web Services offer the technical means to overcome heterogeneity issues in cross-platform communication.

- Web Services offer the power of declaratively specifying the characteristics of message exchanges for the application developer. The corresponding SOAP message exchanges as well as the coupling of these exchanges to application code is supposed to be performed automatically by WS stack implementations.

[6] `http://ws-i.org/`, now moved to OASIS, last access: 12/20/2011

- Web Services technology provides a considerable network effect by being supported on almost arbitrary platforms. So, providing a service as a Web service widens its reach to virtually any client.
- Web Services descriptions, exchange formats, and message contents all are described in XML. This makes the rich set of XML technologies accessible to be applied to Web Services interactions, most notably XML Path Language (XPath) and BPEL.
- Web Services enable the use of Internet protocols such as HTTP or SMTP as transport medium. These are relatively pricey compared to VANs (cf. section 1.2).

The most important drawbacks are:

- Although Web Services technology crucially depends on interoperability, advanced features suffer from substantial interoperability issues [183, 196] and may necessitate realizing QoS properties at the transport layer or manually processing SOAP messages.
- Web Services technology impacts message throughput because using SOAP and XML as message format results in a relatively bulky representation compared to traditional B2Bi technologies such as EDI.
- Web Services technology may affect the maximum size of messages that can be transmitted. Numerous Web Services libraries use Document Object Model (DOM) as underlying processing model for XML which requires loading complete XML trees into main memory. Considering that B2Bi messages may scale up to hundreds of Megabytes this may exceed the capacity of interchange systems.

2.2. Orchestration Technology

Orchestrations are process-based compositions of Web services that govern the local message flow of an integration partner or, more generally, a system component. The purpose of an orchestration process is defining the types of messages that a particular component is able to receive and send, the sequence of these messaging events, and the mapping and processing of data between the messaging events. The term *"orchestration"* (in the area of enterprise computing) was originally coined in [159] where it is put into the context of the choreography-orchestration dichotomy (choreographies will be discussed in the next section). The view taken in [159] focuses on message exchanges of some interaction partners as depicted in figure 2.5 and distinguishes between external and internal Web services calls. The two green boxes on the left-hand and right-hand side of figure 2.5 represent system components that the interaction partners use to couple cross-partner message exchanges to local

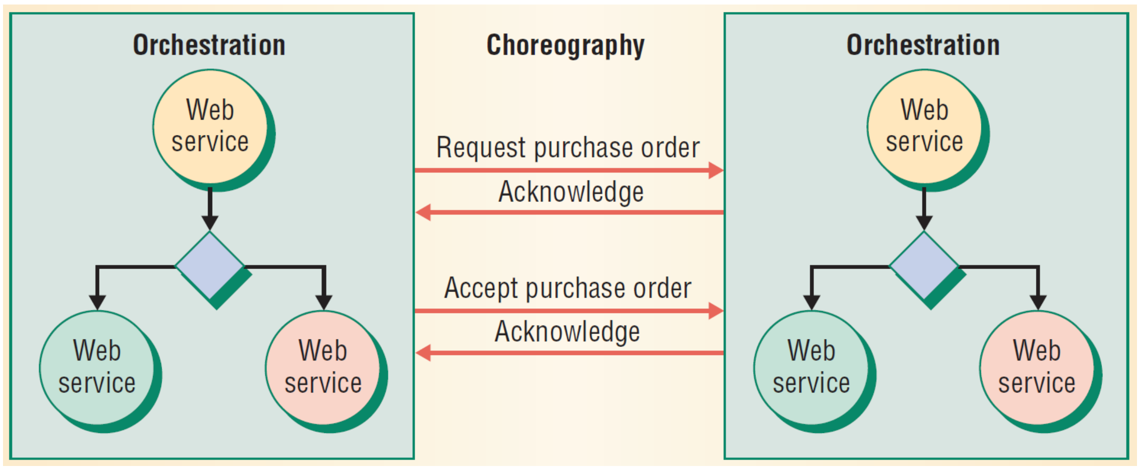


Figure 2.5.: Web Services Choreography and Orchestration, taken from [159]

systems. The arrows between the green boxes represent the cross-partner messages while the interactions with local systems are not visualized. Following this basic paradigm, orchestrations are defined in [159] as follows: *"Orchestration refers to an executable business process that can interact with both internal and external Web services. The interactions occur at the message level"* [159]. This definition fits well with the purpose of this work. However, in order not to unnecessarily constrain the reach of orchestration technology, orchestrations are defined as follows:

> **Orchestration**: An orchestration is the executable definition of the admissible sequences of Web Services-based message exchanges of a system component as well as the data mapping and processing activities between these message exchanges.

Apparently, this definition leaves out the distinction between external and internal message exchanges of [159]. However, this distinction is crucial when using orchestration technology in the area of B2Bi for the implementation of cross-organizational business document exchanges. In such a scenario, orchestrations cover two distinct purposes. Firstly, the specification of admissible interactions with one or more integration partners and, secondly, the coupling of partner message exchanges to internal business applications. From a purely technological perspective, this distinction may seem to be artificial. However, the distinction is crucial from an organizational perspective because the definition of partner interactions needs consent from entities of independent units whereas the interactions with local systems are under the control of one single unit. In order not to break communication links with partners and redo agreement procedures, the definition of partner interactions is supposed to be stable. Conversely, the coupling to local business applications is more flexible as orchestration implementation and business applications are under the same sphere of control. As long as the definition of partner interactions is complied with, it is also possible to optimize the local interactions according to the goals of the respective integration partner. This principle also has been recognized in the BPEL standard that defines the so-called *Abstract Process Profile for Observable Behavior* [137, section 13.3] for capturing the admissible interactions with integration partners. Therefore, an *abstract process* only captures the types and sequences of message exchanges and leaves out a detailed definition of data flow and (possibly) local interactions. *"The main application of this profile is the definition of business process contracts; that is, the behavior followed by one business partner in the context of Web services exchanges"* [137, section 13.3].

This work adopts the concept of distinguishing external and internal interactions as defined in [137] and [159]. In order to reflect the relation of this work to B2Bi the terms *"public orchestration"* and *"private orchestration"* are defined to capture the two different aspects of orchestrations as follows:

> **Public orchestration**: A public orchestration defines the admissible types and sequences of interacting with one particular role of a cross-organizational integration scenario using Web Services. All message exchanges contained in the public orchestration definition are publicly visible, i.e., observable by the partners of the focal role.
>
> **Private orchestration**: A private orchestration is a refinement of a public orchestration that defines the data manipulations as well as interactions with local systems between cross-organizational message exchanges such that the orchestration definition is executable. No valid execution trace of a private orchestration violates the defined message sequences of its corresponding public orchestration.

An important characteristic that distinguishes public and private orchestrations is executability. While private orchestrations explicitly are defined to be executable, public orchestrations are not executable because the data flow perspective is missing. Note further that a role in a cross-organizational integration scenario is not tied to one particular partner link. In a multi-party scenario, an integration partner is characterized by exactly one role that may interact with multiple partners. For characterizing the interaction with one particular integration partner concepts such as *"link role"* or *"link type"* could be introduced. This is basically unnecessary, because the typical integration scenario in B2Bi is defined for exactly two partners anyway (cf. chapter 3). In the end, whether an *"integration role"* is bound to exactly one partner or multiple partners is just a choice of terminology.

The set of proposed orchestration languages has consolidated significantly during the last years. Languages such as the *Web Services Flow Language* (WSFL)[7] or *XLANG*[8] have been given up in favor of BPEL. The development of other languages such as the *Business Process Modeling Language* (BPML)[9] has been discontinued and

[7]`http://xml.coverpages.org/wsfl.html`, last access: 12/20/2011
[8]`http://xml.coverpages.org/xlang.html`, last access: 12/20/2011
[9]`http://xml.coverpages.org/bpml.html`, last access: 12/20/2011

the tooling for some academic approaches like *JOpera* [157] or *XL* [43] do not provide the practical maturity needed for executing processes in an enterprise computing context.

A mature platform for orchestration definition is provided by Windows Workflow[10], but its actual purpose is put more on classical workflow definition in .NET environments than on composition of Web Services. This becomes evident in the fact that a mechanism for exporting a WSDL interface for particular activities is provided, but defining workflows without any Web Services interaction is acceptable as well. This is common for classical workflow languages and therefore a more detailed discussion of these approaches is omitted.

The most widespread standard for public and private orchestration definition is BPEL which builds the basis for executing the binary B2Bi styles of this work and is described next.

2.2.1. Web Services Business Process Execution Language

BPEL is a standard published by the Organization for the Advancement of Structured Information Standards (OASIS) group that covers the process-based composition of Web Services as orchestrations. BPEL relies on WSDL 1.1 for defining the available options for interacting with a particular process. The messages given in WSDL `operations` are bound to *"inbound"* (BPEL constructs `receive`, `onMessage`, `onEvent`, and `pick`) and *"outbound"* (BPEL constructs `invoke` and `reply`) message activities. The binding between WSDL messages and BPEL message activities relies on the concept of BPEL `partnerLinks` that take the role of the provider or consumer of WSDL `portTypes`. Once these `partnerLinks` are defined, message activities can be bound to the `operations` of a `portType`. The first step in defining `partnerLinks` is the definition of a so-called `partnerLinkType` within a WSDL file that is used to give a role name to one or two portTypes, i.e., for defining unidirectional or bidirectional link types. In a BPEL process definition, a `partnerLink` element then is used to associate the focal process with one of those role names. For specifying that the focal BPEL process is the provider of the `portType` implied by a `partnerLinkType` role, the respective `partnerLink` element carries a `myRole` attribute the value of which is the role name. Conversely, if the focal BPEL process is proposed to consume a corresponding `portType` then a `partnerRole` attribute is used. As example, consider the sample BPEL process *'purchaseOrderProcess'* depicted in figure 2.6 that defines one unidirectional (*'purchasing'*) and three bidirectional `partnerLinks` (*'invoicing'*, *'shipping'*, and *'scheduling'*). For characterizing each communication direction of each `partnerLink` a separate WSDL `portType` is used (*'purchaseOrderPT'*, *'computPricePT'*, *'invoiceCallbackPT'* etc.). In listing 2.1, the `partnerLinkType` element is used as described above for defining the undirectional link type *'purchasingLT'* and the bidirectional link type *'invoicingLT'*.

[10]`http://msdn.microsoft.com/en-us/netframework/aa663328`, last access: 12/20/2011

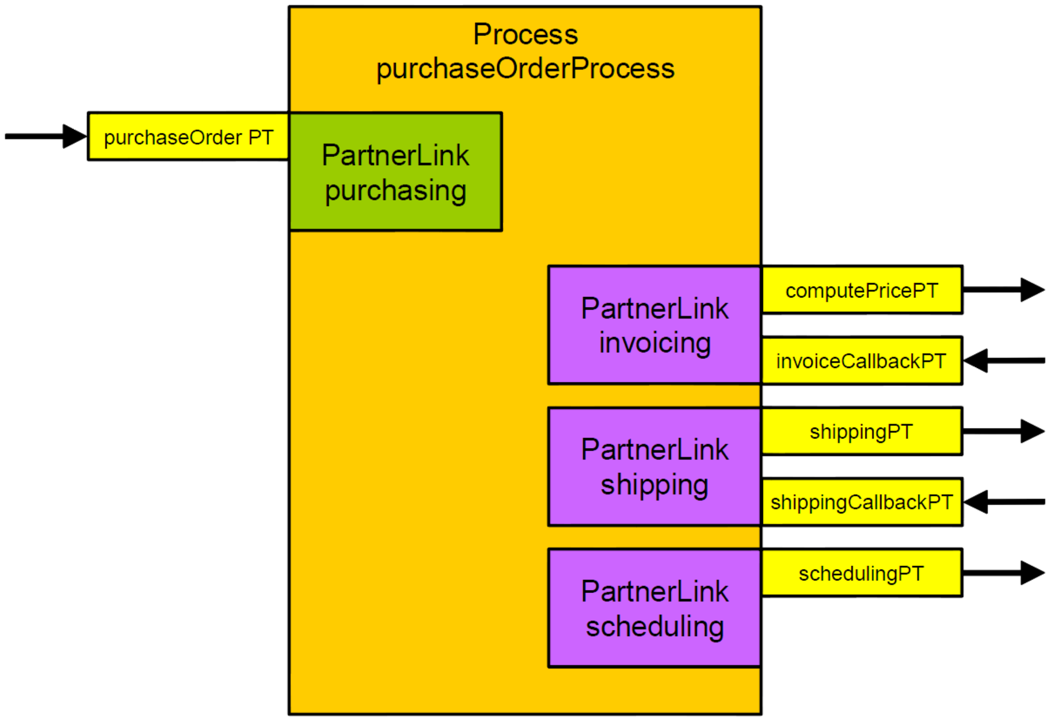


Figure 2.6.: Conceptual Visualization of a BPEL Process with PartnerLinks, taken from [137]

Listing 2.1: Definition of `PartnerLinkType` in a WSDL File, taken from [137]

```
<plnk:partnerLinkType name="purchasingLT">
  <plnk:role name="purchaseService" portType="pos:purchaseOrderPT"/>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="invoicingLT">
  <plnk:role name="invoiceService" portType="pos:computePricePT"/>
  <plnk:role name="invoiceRequester" portType="pos:invoiceCallbackPT"/>
</plnk:partnerLinkType>
```

The definition of the `partnerLinkTypes` in the corresponding WSDL file is the basis for defining the actual `partnerLinks` in the BPEL process definition as exemplified in listing 2.2. In order to define the provision of the `operations` of `portType` *'purchaseOrderPT'* (listing 2.1) the `partnerLink` *'purchasing'* in line 2 of listing 2.2 first references the `partnerLinkType` *'purchasingLT'* and then references the link type's *'purchaseService'* role using the `myRole` attribute. Similarly, the `partnerLink` *'invoicing'* in the next line uses the `myRole` and `partnerRole` attributes to define the provision of the *'invoiceCallbackPT'* and the consumption of the *'computePricePT'* `portTypes`. Although not shown in listings 2.1 and 2.2, the provision of unidirectional `partnerLinks` for consumption of `operations` only is possible as well.

Listing 2.2: Definition of `PartnerLink` in a BPEL Process, taken from [137]

```
1 <partnerLink name="purchasing"
2   partnerLinkType="lns:purchasingLT" myRole="purchaseService"/>
3 <partnerLink name="invoicing" partnerLinkType="lns:invoicingLT"
4   myRole="invoiceRequester" partnerRole="invoiceService"/>
```

The definition of several `partnerLinks` for the process in figure 2.6 also shows that more than one WSDL file can be used to define the available interactions with a BPEL process. This implies that a BPEL process can play several roles. In that regard, note that a BPEL role defined by a WSDL `portType` is just a software engineering level role and may or may not completely represent the role of an integration partner within an integration scenario as described above.

While `partnerLinks` are sufficient to define the available types of interaction with a BPEL process, the concept of *"process correlation"* is needed to distinguish between different instances of the same BPEL process definition. BPEL offers so-called `correlationSets` that can be interpreted as identifiers of a process instance. An identifier, in turn, is a set of correlation `properties` that commonly determine process instances. However, a single `property` is typically used in practice. This `property` then is bound to the contents of WSDL `message` definitions using so-called `propertyAliases` that are included in WSDL files. A `propertyAlias` conceptually is a rule for extracting correlation `properties` from WSDL messages. Listing 2.3 shows an example for how the correlation `property` *'processId'* is supposed to be extracted from a *'poStatusQuery'* message that may be defined within the *'purchaseOrderPT'* portType above. To do so, the `part` attribute is first used to identify the WSDL `message part`. Then, an XPath `query` is used for capturing the XML element that carries the content to be used as process identifier. At run-time, the BPEL engine that executes the *'purchaseOrderProcess'* then is in charge of applying the `propertyAlias` extraction rule upon receipt of a *'poStatusQuery'* message and to route it to the corresponding process instance.

Listing 2.3: Sample `PropertyAlias` for Defining Process Identification Data

```
<vprop:propertyAlias propertyName="corr:processId" messageType="tns:poStatusQuery"
    part="poStatusQueryPart">
  <vprop:query>proc:ProcessData/proc:PurchaseId</vprop:query>
</vprop:propertyAlias>
```

The alternative to using `correlationSets` for identifying process instances is leaving the problem to proprietary functionality of BPEL engines which typically relies on some non-standardized SOAP header extensions. However, using proprietary functionality is not acceptable in a B2Bi-setting.

The actual control flow definition of a BPEL process relies on scopes and control flow activities. The most common purpose of `scopes` is the delineation of the visibility of `variables` as well as fault and event handling routines which are defined using `faultHandlers` and `eventHandlers` elements, respectively. `Scopes` can be nested and the first (implicit) scope is defined by the root BPEL element `process`. Within each `scope`, a series of control flow constructs like `sequence`, `if`, `while` and `pick` are used to determine the order of message activities. In addition, the BPEL `assign` activity can be included in the order of activities for the purpose of mapping data between the process variables. XPath and Extensible Stylesheet Language Transformations (XSLT) are available for data manipulation within `assign` activities.

The control flow definition of BPEL is basically block-structured which means that control flow structures are hierarchically nested (note that this is not an issue that results from nesting XML tags). This concept may be problematic if arbitrary graph-like structures are supposed to be implemented [76]. For example, consider

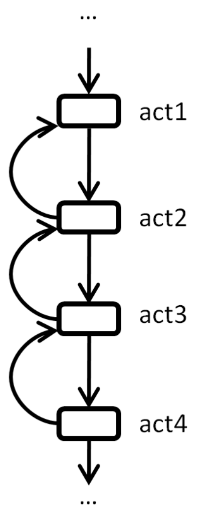


Figure 2.7.: Sample Irreducible Loop Structure

the graph-based loop structure depicted in figure 2.7 that contains four activities *'act1'* to *'act4'* and loops between *'act1'* and *'act2'*, *'act2'* and *'act3'*, and *'act3'* and *'act4'* respectively. Obviously, execution traces that include the alternating repetition of the activities of these loops are possible. As the loops overlap in nodes *'act2'* and *'act3'*, execution traces also may include more complicated loop sequences such as repetitions of *'act1,act2,act3,act2'*, but this is of minor importance for the example. Now look at figure 2.8 that contains an attempt to represent the loop of figure 2.7 in a block-structured manner. Each rectangle represents a BPEL control flow element and nesting of rectangles corresponds to nesting of control flow elements. Beyond that, the execution is performed top-down. This means that the `while` loop *'while1'* repeats `sequence` *'seq1'* where first `while` loop *'while2'* is performed and once *'while2'* has terminated loop *'while3'* is performed. Sequences *'seq2'* and *'seq3'* as well as (arbitrary) activities *'act1'* to *'act4'* are executed correspondingly. Loop *'while2'* is able to produce alternating repetitions of *'act1'* and *'act2'* and loop *'while3'* is able to produce alternating repetitions of *'act3'* and *'act4'*. Furthermore, loop *'while1'* is able to produce alternating repetitions of *'while2'* and *'while3'*. However, this does not include the alternating repetition of *'act2'* and *'act3'* which is possible in the graph-based loop of figure 2.7. Including another BPEL `while` loop between *'while2'* and *'while3'* that contains *'act2'* and *'act3'* is not valid either because then a trace *'act1,act2,act2,act3'* is possible that is not part of the execution traces of figure 2.7. The problem boils down to the basic issue that alternating repetitions of *'act1'* and *'act2'* as well as *'act2'* and *'act3'* require the use of two BPEL `while` loops that overlap in *'act2'*. However, as block-structures forbid overlapping elements, *'act2'*

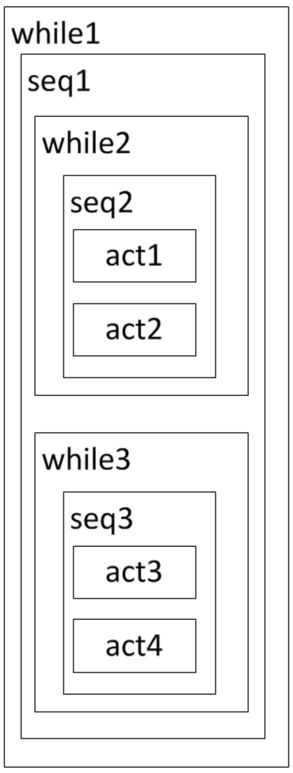


Figure 2.8.: Invalid Representation of Sample Loop

must be duplicated which is not valid either. Of course, the problem could be solved by introducing auxiliary variables and `if` conditions for checking whether another execution of some activity is eligible. Consider that control flags could be set after performing *'act3'* that exclude *'act4'* from the current iteration of *'while3'* and *'act1'* from the next iteration of *'while2'*. Then, performing *'act2'* after *'act3'* would be possible. Yet, such an approach relying on auxiliary variables corresponds to hand-coding the control flow structure and is not a true block-structured representation of the problem.

Note that not all BPEL elements are strictly block-structured in nature. BPEL `onEvent` elements defined in the `eventHandlers` section of a `scope` can be used to receive and process a message in parallel to the main execution thread of the scope. By attaching multiple `onEvents` to a scope and including calls to other `onEvents` in the event processing logic, i.e., having the BPEL process call itself from within `onEvent` blocks, an arbitrary graph structure could be created (cf. [152]). However, this concept of having a BPEL process invoke itself is not well-supported by current BPEL engines. Moreover, a major drawback of this solution is performance impact because it necessitates passing a message through the Web Services stack for the only purpose of passing on control flow. A more direct way of offering graph-like control flow without auxiliary variables is provided by the BPEL `flow` element that is used to create parallel structures. Between two activities of different branches of a `flow` element, BPEL `links` can be introduced. The semantics is such that the source activity of the `link` must be performed before the target activity. By arranging all activities of a process in the branches of a `flow` element and then inserting `links` correspondingly, graph-based structures can be created. Therefore, [81] find that BPEL is not block-structured. However, this concept of representing graph-like structures in BPEL is very limited because no cycles are allowed by the standard: *"A <link> declared in a <flow> MUST NOT create a control cycle, that is, the source activity must not have the target activity as a logically preceding activity. This implies that such directed graphs are always acyclic"* [137, section 11.6.1]. So, the claim that BPEL is a predominately block-structured language is retained in this work.

Note that this quick overview of BPEL is by far not complete and covers the most important elements that are used in this work only. If necessary, some more detailed or use case specific information is provided throughout this work. For a comprehensive treatment, the reader is referred to the BPEL standard [137].

2.3. Choreography Technology

Choreography is the second part in the orchestration-choreography dichotomy coined by Chris Peltz in 2003 [159]. The last section defined orchestrations as the executable definition of the admissible Web Services-based message exchange sequences of an integration partner. Figure 2.5 (last section) visualizes the interaction between two orchestrations and identifies the message exchanges between the orchestrations as *"choreography"*. This concept is paraphrased by Peltz as follows: *"Choreography tracks the message sequences among multiple parties and sources-typically the public message exchanges that occur between Web services- rather than a specific business process that a single party executes"* [159]. While this ties the notion of *"choreography"* to Web Services, today, there are a number of Web Services agnostic choreography languages such as UMM [208], ebBP [134] or Let's Dance [245]. Notably, the authors of the respective languages all claim to provide a choreography language. Capturing publicly visible messages between entities has remained as common characteristic of choreography languages. In this work, the idea of not tying the term *"choreography"* to Web Services is followed because a choreography can be interpreted as an abstract protocol that constrains the implementation of orchestrations. There is no reason why a protocol definition should limit possible implementations in terms of technology. Moreover, tying the term to Web Services would exclude a large number of languages that have been used for protocol definition beforehand. Therefore, the term *"choreography"* is defined for the purpose of this work as follows:

> **(Global) Choreography**: A (global) choreography is the characterization of the admissible types and sequences of message exchanges between the partners of a cross-organizational integration scenario. This implies that all message exchanges are visible to at least two partners.

Although not explicitly stated, the main purpose of defining choreographies is facilitating *"behavioral interoperability"* between interacting parties (as opposed to *messaging interoperability* discussed in section 2.1). So, if partner A is obliged to first send message Y to B and then message Z, partner B should be obliged to first receive Y and then Z correspondingly. The exact format of specifying such constraints depends on the application purpose, but it can be observed that a global choreography definition also may be projected to the local obligations of one particular role. Therefore, some researchers make the distinction between global and local choreographies [59] where a *"global choreography"* refers to the overall contract and *"local choreography"* to the obligations of one single role. Therefore the term *"local choreography"* is defined as follows:

> **Local Choreography**: A local choreography defines the admissible types and sequences of interacting with one particular role of a cross-organizational integration scenario. All message exchanges contained in the public orchestration definition are publicly visible, i.e., observable by the partners of the focal role.

The attentive reader may have noticed that the only difference to the definition of *"public orchestration"* is the missing reference to Web Services. Indeed, local choreographies and public orchestrations serve the same purpose. The question just is whether the local communication obligations of a particular role should be captured more abstractly or in a Web Services specific manner. As this work primarily takes a global view on choreographies, the term *"choreography"* if used without further information always refers to a global choreography.

Specifying the message exchanges between integration partners or, more abstractly, system components is a common task in computing which calls for a categorization of the concept. Decker, Kopp and Barros [29] developed a categorization of choreographies based on two pivotal properties of choreography languages. Firstly, they distinguish between *"interconnection"* choreographies and *"interaction"* choreographies. Interconnection choreographies focus on the local send and receive actions of individual partners as well as the interconnection of corresponding send/receive actions. Conversely, *interaction* choreographies treat corresponding send and receive events as atomic actions and define sequences of these actions. Secondly, they distinguish between *implementation specific* choreographies that capture implementation level concepts like communication technology (say, Web Services) and *implementation independent* choreographies that are agnostic to those concepts.

While this categorization for sure is pivotal it still captures languages with considerable differences in the same category. For example, Let's Dance and ebBP can both be characterized as implementation-independent interaction choreographies. However, ebBP aims at specifying the business document exchanges between enterprises while Let's Dance targets supporting service interaction patterns [11] with a visual choreography language. Although these two goals overlap, they result in substantially different concepts. ebBP provides support for referencing existing business document libraries as provided by B2Bi communities like RosettaNet and for specifying security and reliability requirements. Moreover, it assumes a protocol consisting of several message exchanges for implementing a business document exchange. Let's Dance, in turn, offers functionality for analyzing such protocols and provides a rich set of features for modeling service interactions.

These differences are a first hint that a refined choreography notion is needed. This question was investigated in [179] as part of this dissertation project and the analysis framework for conceptual modeling languages proposed by Wand and Weber [229] was used to derive *B2Bi/Services/Conceptual Choreographies* as distinct choreography classes that are largely orthogonal to the categorization presented in [29].

The framework of [229] offers the components *task factors* and *modeling grammar* that are relevant for establishing categorizations of languages. Task factors capture the purpose of using a language whereas *modeling grammar* captures the constructs and rules for creating models. While choreography languages may lend itself to a variety of different purposes, it is striking that almost all choreography languages and approaches underline their relevance for B2Bi. Publications such as [37] and [187] that analyze the development phases of B2Bi and therefore are suited to identify *task factors* reveal that choreography technologies typically are used to fill the semantic gap between business process models and orchestration models. This can be done by refining business process model concepts or by abstracting orchestration model concepts. For example, BPEL4Chor [31] reuses a considerable part of BPEL concepts which corresponds to *abstracting orchestration model concepts.* Conversely, ebBP uses so-called BusinessTransactions to specify requirements of message exchanges which corresponds to *refining the business process model.* Finally, for some choreography languages it is not easily decidable whether they are semantically more close to the business process layer or to the orchestration layer. For example, IOWF-Nets [214] capture choreographies as interconnected Petri Nets. This resembles the concept of composing a choreography by connecting orchestrations and therefore seems to imply a close relationship to the orchestration layer. However, the business process layer may contain partner-local models as well and as IOWF-Nets do not have technology specific concepts, they could potentially be used for analyzing the business process layer itself.

These differences are also reflected in the core building blocks of the various choreography languages (cf. *modeling grammar* [229]). In BPEL4Chor, *communication activities* are used to capture the send and receive events of the individual partners. The WSDL operation types `one-way` and `request-response` are adopted to allow for *"higher similarity between participant behavior descriptions and orchestrations"* [32, section 4.2]. So, although BPEL4Chor is defined such that it does not *technically depend* on WSDL (by removing the *partnerLink*, *portType*, and *operation* attributes from BPEL message activities), it can be concluded that BPEL4Chor is *designed for* service based interactions. In ebBP, a BusinessTransaction represents a B2Bi domain specific configuration of a business document exchange with B2Bi parameters for a lower-level execution protocol. Finally, the core building blocks of languages like IOWF-Nets or Let's Dance neither rely on Web Services concepts nor define B2Bi domain concepts.

On the basis of this analysis of *tasks factors* and *modeling grammar* as supposed by [229], the three different classes of choreography languages (largely orthogonal to the categorization in [29]) used in the introduction of this work can be identified (cf. section 1.2):

- **B2Bi Choreographies** that offer B2Bi specific concepts like configurable BusinessTransactions and which semantically are close to business process models.
- **Services Choreographies** that offer Web Services technology specific concepts and are close to the orchestration layer.
- **Conceptual choreographies** that offer concepts driven by the purpose of analysis and may be used to complement/analyze the business process layer as well as the orchestration layer.

There is a series of criteria that discriminate between these choreography classes available in [179], but the selection of the choreography representation formats for this work is aligned with the B2Bi requirements introduced in the next chapter. So, instead of describing a comparison of choreography classes that is not aligned with the purpose of this work the reader is referred to [179]. As core technologies used in this work, ebBP and BPMN are introduced in sections 2.3.1 and 2.3.2, respectively. Sections 2.3.3, 2.3.4, 2.3.5 then sketch the most relevant representatives of the above choreography classes that may be suitable as alternatives to the technologies chosen.

2.3.1. ebXML Business Process Specification Schema

The initial version 1 of the XML format ebBP had its roots in UMM and RosettaNet standards and was proposed as one of the results of the ebXML initiative. The current version of ebBP [134] was released in 2006 and did not dramatically change the base concepts. The most important new feature of ebBP version 2 probably is the possibility of defining multi-party choreographies.

Basically, ebBP is an interaction-style, implementation-agnostic B2Bi choreography language that is centered around the concept of business document exchanges. An ebBP choreography is denoted as *"BusinessCollaboration (BC)"* in ebBP terminology and characterized in the standard as follows: *"A Business Collaboration consists of a set of roles that collaborate by exchanging Business Documents through a set of choreographed transactions"* [134, line 436/437].

The core building block of BCs for specifying the exchange of business documents are so-called BusinessTransactionActivitys (BTAs). A BTA specifies the execution of a particular BusinessTransaction (BT) definition at a particular point of the choreography, i.e., the same BT definition can be used several times within one choreography. In turn, a BT is the technology-agnostic specification of the exchange of a business document and an optional business document response between exactly two partners. Depending on the number of business documents exchanged, BTs also are denoted as *"One-"* or *"Two-Action-BTs"* or as *"One-"* or *"Two-Way-BTs"*. The sender of the request document is referred to as the *"requester role"* or *"requesting role"* and the receiver of the request document is referred to as the *"responder role"* or *"responding role"*. Whether or not a response document is exchanged within a BT is irrelevant for the denotation of those names. For the exchange of each business document of a BT, so-called BusinessActivities (BAs) are used to specify the business document to be exchanged together with additional business signals and business related QoS properties. Business signals, namely ReceiptAcknowledgements (RAs) and AcceptanceAcknowledgements (AAs), are designed to give the business document sender information about the state of a business document's processing by the receiver. The purpose of a RA is characterized as follows: *"The Receipt Acknowledgement Business Signal, if used, signals that a message (Request or Response) has been properly received by the BSI [business service interface] software component. The property isIntelligibleCheckRequired allows partners to agree that a Receipt Acknowledgement SHOULD confirm a message only if it is also legible. Legible means that it has passed structure/schema validity check"* [134, lines 1276-1280]. The term *"ReceiptAcknowledgement"* is actually a misnomer because real acknowledgment of receipt requires the use of reliable messaging protocols (see section 2.1) or distributed commit protocols. Instead, the RA refers to rather static validation procedures that not necessarily require involvement of business applications for querying master or process data. Conversely, sending AAs barely is conceivable without prior involvement of business applications: *"The Acceptance Acknowledgement Business Signal, if used, signals that the message received (Request or Response) has been accepted for business processing and that processing is complete and successful by the receiving application, service or a receiving business application proxy"* [134, lines 1292-1295]. However, note that an AA does not imply business level acceptance of a particular business document such as a purchase order. It merely signals that the receiver of the document will be able to process it with a very high probability. Actual acceptance at the business level, e.g., a purchase order confirmation, requires the exchange of another business document or the implicit convention that every valid purchase order will be accepted. In addition to the RA and AA business signals, corresponding ReceiptAcknowledgementException (RAE) and AcceptanceAcknowledgementException (AAE) exception signals are defined that signal validation errors. For exceptions beyond RA and AA validation errors, the GeneralException (GE) exception signal is defined [134, section 3.6.2.3].

According to the terminology of section 2.1, the exchange of business documents as well as business signals all necessarily represent application level message exchanges. Further QoS attributes are available for configuring BTs that may be implemented at the messaging or transport level that are listed in table 2.1. Each attribute is associated with its object of specification, i.e., whether it applies to the business document (Doc), the BA, BT or BTA. Most of the ebBP QoS attributes are self-explanatory, but *hasLegalIntent* and *isConcurrent* deserve additional explanation. For *"hasLegalIntent"*, ebBP states that *"The hasLegalIntent attribute could have* *widely differing interpretations and enforceability depending on type of business, process, and jurisdiction"* ([134, section 3.4.9.7]). *"isConcurrent"* (cf. [134, section 3.4.10.1]) specifies whether multiple instances of a BT within the same process or in different processes (with the same party) are allowed to be active at the same time.

QoS Attribute	Level of Specification
isAuthenticated	Doc
isConfidential	Doc
isTamperDetectable	Doc
isIntelligibleCheckRequired	BA
isNonRepudiationRequired	BA
isNonRepudiationReceiptRequired	BA
timeToAcknowledgeReceipt	BA
timeToAcknowledgeAcceptance	BA
isAuthorizationRequired	BA
retryCount	BA
isGuaranteedDeliveryRequired	BT
hasLegalIntent	BTA
isConcurrent	BTA
timeToPerform	BTA

Table 2.1.: ebBP QoS Attributes and Specification Levels

Together, the selection of business documents, business signals and QoS attributes makes up a *"BT configuration"*. The ebBP standard offers so-called *"business transaction patterns"* (originally defined in UMM [207]) as associated concept that allows for alignment with different business scenarios. For example, the *"Commercial Transaction"* pattern describes a Two-Way-BT that establishes a *"formal obligation between parties"* [134, lines 1162-1163] and the *"Information Distribution"* pattern *"represents an informal information exchange between parties"* [134, lines 1173-1174]. Technically, the different patterns represent different instantiations of a BT configuration. For example, a *"Commercial Transaction"* defines the exchange of two business documents and explicitly requires the exchange of a RA whereas an *"Information Distribution"* defines the exchange of one business document and defines the exchange of a RA as optional. One important benefit of those patterns therefore is a guideline in choosing a BT configuration that fits the application purpose. The *"DataExchange"* pattern is special in allowing for arbitrary changes of the BT configuration and therefore can be interpreted to cover all other patterns in a technical sense. This pattern is provided by the ebBP specification in order to allow for partner specific exchange semantics.

Listing 2.4 shows the ebBP format for a BT configuration that includes all possible configuration options for a One-Way-BT. The example shows the specification of the exchange of a purchase order confirmation (POC) using a *"DataExchange"* business transaction type and the referenced business document is taken from the RosettaNet business document library[11]. Alternative configurations could easily be derived by omitting XML attributes or elements that represent business signals or QoS attributes. Section 4.3 will show how a BT configuration can be represented as a pair of protocol machines and section 5.2 will cover the actual implementation.

Listing 2.4: RosettaNet-Based BT Example

```
<DataExchange
 name="bt-PIP3A20"
 nameID="bt-PIP3A20"
 isGuaranteedDeliveryRequired="true">
 <RequestingRole name="POC sender"
  nameID="PIP3A20-role-sender"/>
 <RespondingRole name="POC receiver"
  nameID="PIP3A20-role-receiver"/>
 <RequestingBusinessActivity
  name="Send POC"
  nameID="PIP3A20-ba-req"
  isIntelligibleCheckRequired="true"
  isNonRepudiationRequired="true"
  isNonRepudiationReceiptRequired="true"
  retryCount="3"
  timeToAcknowledgeReceipt="PT3M"
  timeToAcknowledgeAcceptance="PT6M">
  <DocumentEnvelope
   name="PIP3A20 POC"
   businessDocumentRef="PIP3A20-POC"
   nameID="PIP3A20-POC-de"
   isAuthenticated="transient"
   isConfidential="transient"
   isTamperDetectable="transient"/>
  <ReceiptAcknowledgement
   name="ra" nameID="PIP3A20-ra"
   signalDefinitionRef="ra2"/>
  <ReceiptAcknowledgementException
   name="rae" nameID="PIP3A20-rae"
   signalDefinitionRef="rae2"/>
  <AcceptanceAcknowledgement
   name="aa" nameID="PIP3A20-aa"
   signalDefinitionRef="aa2"/>
  <AcceptanceAcknowledgementException
   name="aae" nameID="PIP3A20-aae"
   signalDefinitionRef="aae2"/>
 </RequestingBusinessActivity>
</DataExchange>
```

Beyond BTAs, CollaborationActivities (BCAs) are available as building blocks of a BC. A BCA defines the execution of a BC within another BC and thus enables hierarchical decomposition of choreographies. Beyond providing a construct for defining the point in control flow at which a BT or BC is supposed to be performed, BTAs and BCAs are used to define *"role mapping"*. Role mapping defines the association between the roles of BCs and the roles of BTAs and BCAs of the respective BC. For example, assume that a particular BC defines the roles *bc-vendor*

[11]`http://www.rosettanet.org/Standards/RosettaNetStandards/PIPDirectory/tabid/476/Default.aspx`, last access: 12/20/2011

and *bc-purchaser* and that some BT defines the exchange of a purchase order. Then, ebBP's BTA construct can be used to bind the *bc-vendor* to the requesting role of the BT and the *bc-purchaser* to the responding role. If it is necessary to send a purchase order in the opposite direction at a later point in time of the BC then another BTA can be used to bind the *bc-vendor* to the responding role and the *bc-purchaser* to the requesting role. In addition, so-called TimeToPerform (TTP) values are available for the specification of BTA and BCA timeouts.

The control flow structure of a BC is an almost arbitrary graph that uses `Forks`, `Joins`, `Decisions` and `Transitions` to describe the flow between BTAs and BCAs. ebBP `Forks` can be of type `OR` or `XOR` and ebBP `Joins` carry a boolean `waitForAll` attribute. In this work, `Joins` with `waitForAll` set to true are denoted as *"AND-Joins"* and `Joins` with `waitForAll` set to false are denoted as *"OR-Joins"*. `Forks` of type `XOR` (*"XOR-Fork"*) are allowed to trigger exactly one successor whereas an arbitrary number of successors can be triggered by a `Fork` of type `OR` (*"OR-Fork"*). If an OR-Fork is matched by an AND-Join then all Fork successors have to be triggered [134]. The `Fork` of such a combination will be denoted as *"AND-Fork"*. If an OR-Fork is matched by an OR-Join then an arbitrary number of `Fork` successors may be triggered, but the behavior of the OR-Join is not precisely defined. ebBP `Decisions` connect multiple BTAs/BCAs and choose between different paths by assigning guards to the corresponding links whereas ebBP `Transitions` connect exactly two BTAs/BCAs and also allow for the specification of guards.

The guards that are available for routing the control flow of ebBP choreographies are either predefined protocol outcomes or expressions on the exchanged business documents. Generic protocol outcomes are captured using ebBP's so-called ConditionGuardValue (CGV) language. A CGV expression refers to typical results of a BT execution and defines `Success` and `Failure` as general results. The available failure results are depicted in figure 2.9. The expression `AnyProtocolFailure` is used to capture a series of protocol failures such as a response document or business signal that is not provided on time (`ResponseTimeout` or `SignalTimeout`) or validation errors (e.g., a `RequestReceiptFailure`). If the result is not an `AnyProtocolFailure` from a protocol perspective then it is a `ProtocolSuccess`. In addition to the protocol perspective, the type (not the content) of the business document exchanged may be considered in CGV expressions. If the BT execution was executed successfully from a protocol perspective and a business document type (as agreed by the integration partners) that indicates failure was exchanged, then the result is a `BusinessFailure`. Conversely, if the business document type was agreed to represent success then the CGV result expression is `BusinessSuccess`. Interestingly, this leads to the situation that a `BusinessFailure` is considered to be a failure although it implies `ProtocolSuccess`. Moreover, a result is not a generic result if it requires use case specific agreements of integration partners such as the definition of business document types that indicate failure or success. In consequence, only the CGV expressions that refer to protocol outcomes are used in this work.

If a BT execution was successful (`ProtocolSuccess`) then the evaluation of the business document contents may be sensible. ebBP allows for several languages for evaluating document contents, in particular XPath versions 1 and 2, XSLT versions 1 and 2, CAM version 1, and XQuery version 1. Moreover, user-defined expression languages can be defined as declared on page 65: *"This specification does not limit the type and number of languages a BSI [business service interface] MAY support for variables or condition expressions"* [134, page 65]. Note that ebBP does not define a default expression language for capturing the results of BC executions.

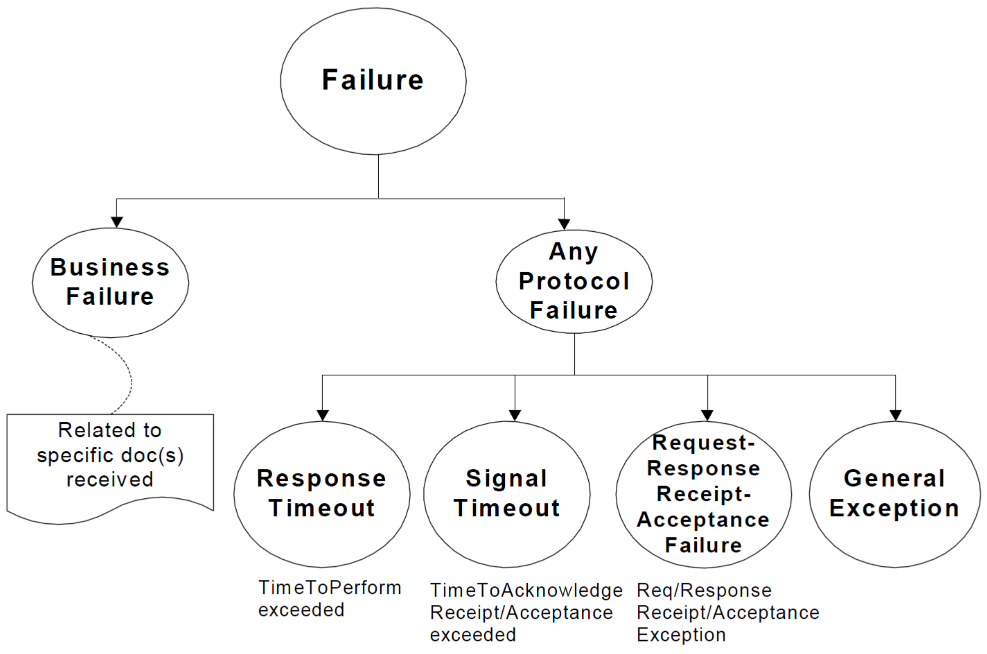

Failure
Business Failure
Any Protocol Failure
Related to specific doc(s) received
Response Timeout
Signal Timeout
Request-Response Receipt-Acceptance Failure
General Exception
TimeToPerform exceeded
TimeToAcknowledge Receipt/Acceptance exceeded
Req/Response Receipt/Acceptance Exception

Figure 2.9.: Generic Protocol Failure Results of a BT, taken from [134, page 85]

Again, this section just provides an overview of the most important ebBP constructs that are relevant for this work. For a comprehensive introduction, the reader is referred to the standard [134].

2.3.2. BPMN Choreographies

The BPMN choreography standard was released as a visual format for choreography specification as new part of the BPMN 2.0 specification [150]. BPMN defines an XML serialization format, but its purpose is rather diagram interchange than modeling.

The basic building block of BPMN choreographies are so-called *"choreography tasks"* that *"represent[] an Interaction, which is one or two Message exchanges between two Participants"* [150, section 11.4.1]. In addition, *"sub-choreographies"* and *"call choreographies"* can be used for hierarchical decomposition and a wealth of BPMN *"gateways"* and *"events"* can be used for routing the control flow between these interactions. In so far, the basic underlying paradigm of BPMN choreographies is very similar to ebBP choreographies. Consistently, BPMN choreographies can be classified as interaction-centric choreographies. As the modeler is not forced to include implementation technology specific artifacts, BPMN choreographies can further be classified as technology agnostic.

The classification according to the three classes *B2Bi choreographies*, *Services choreographies* and *Conceptual Choreographies* (cf. above) is not as easy.

Conceptually, BPMN choreographies are close to the Business Process Management (BPM) layer due to its abstract business-oriented perspective on processes. However, for the classification as B2Bi choreography, B2Bi concepts such as an explicit construct for representing business documents or common B2Bi QoS parameters as defined in table 2.1 are missing.

Some other elements indicate a semantical relation to services choreographies. The introduction of the BPMN choreography chapter itself [150, section 11] indicates that support for service interaction patterns [11] is a goal. Several constructs such as choreography task markers for looping, parallel multi-instance or sequential multi-instance execution as well as multi-instance markers for participants [150, section 11.4.1] also suggest that support for service interactions patterns was a design goal of BPMN choreographies (cf. [32]). In addition, BPMN allows for deriving WSDL operations for message exchanges (even if a visual syntax is not provided for that). Note, however, that the user is deliberately not required to bind all kind of BPMN constructs to Web Services concepts in order to retain implementation technology independence. So, BPMN is not a true services choreography language either.

From a technological perspective, BPMN choreographies are neither a true B2Bi choreography language nor a services choreography language. Yet, it can be adapted to either use which is a typical indicator for conceptual choreographies. Hence, BPMN choreographies are classified as conceptual choreographies that are used for system specification rather than for formal analysis.

All BPMN choreography elements that are relevant for this work will be covered in detail in chapter 6 where BPMN will be used and adapted for visualizing this work's B2Bi choreography styles. Therefore, a more detailed description of BPMN choreographies is not provided in this section.

2.3.3. Alternative B2Bi Choreography Languages

The most important alternative B2Bi choreography language is the UN/CEFACT Modeling Methodology (UMM) which is a standard of the United Nations' Centre for Trade Facilitation and E-business (UN/CEFACT) and is defined as a Unified Modeling Language (UML) profile for modeling B2Bi scenarios. UMM has been developed since (at least) 2001 [207] and has been updated to version 1 [208] in 2006 and version 2 [210] in 2011. The latest version of UMM adds considerable enhancements to its predecessors. These include support for UML 2, technical improvements in terms of business document definition or explicit modeling of alternative document responses, and a reorganization of the various views on B2Bi scenarios.

The integration of several views on B2Bi scenarios is a conceptual strength of UMM as it allows for separation of concern and reciprocal cross-checks between the views, i.e., the contents of one view may be used for checking the contents of another view for completeness or for validation purposes. The *Business Requirements View* for capturing requirements of B2Bi scenarios, the *Business Choreography View* for specifying the actual message exchanges, and the *Business Information View* for describing the exchanged documents are the three main views of UMM. Out of these three views, the `Business Choreography View` provides the actual choreography language of UMM. Again, this view is split up into three different views, i.e., the *Business Transaction View* for modeling ebBP-like *business transactions* as activity diagrams, the *Business Collaboration View* for modeling ebBP-like *business collaborations* as activity diagrams (also), and the *Business Realization View* for assigning business partners to collaboration roles. Figure 2.10 gives an overview of UMM's views on B2Bi scenarios. The concepts of UMM's Business Choreography View and ebBP are very similar which is due to the common roots in the ebXML project[12]. Indeed, the semantic similarity is so high that UMM's Business Choreography View and ebBP can be interpreted as UML format and XML format of the same concepts (see [60] for an in-depth comparison of early UMM and ebBP versions). Consistently, the choreography part of UMM can be classified as interaction-centric and implementation independent B2Bi choreography language.

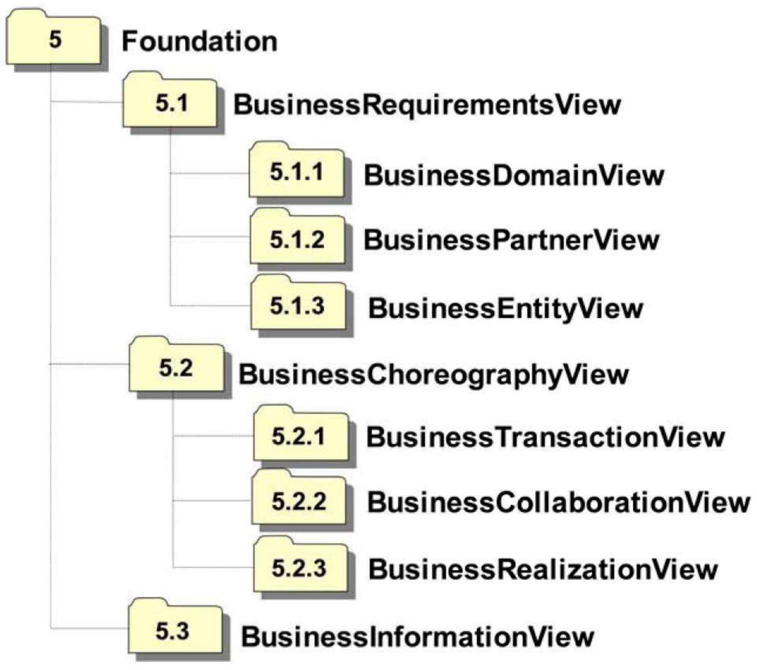

5 Foundation
5.1 BusinessRequirementsView
5.1.1 BusinessDomainView
5.1.2 BusinessPartnerView
5.1.3 BusinessEntityView
5.2 BusinessChoreographyView
5.2.1 BusinessTransactionView
5.2.2 BusinessCollaborationView
5.2.3 BusinessRealizationView
5.3 BusinessInformationView

Figure 2.10.: UMM's Views on B2Bi Scenarios, taken from [210]

Despite the conceptual benefits of multi-view modeling approaches, UMM is sometimes criticized by practitioners for being too complex due to the number of views

[12]`http://www.ebxml.org/`, last access: 12/20/2011

that are supposed to be created. The Business Choreography Language (BCL) [248] addresses this critique by merging the most important artifacts of the different UMM views into a domain specific language that allows for modeling choreographies in one single diagram. As BCL diagrams are aligned with UMM concepts, BCL can be classified as interaction-centric and implementation independent B2Bi choreography language, too. Although BCL is not an official standard like UMM, its potential use is providing a simplified interface to UMM artifacts and thus enabling the participation of integration partners in B2Bi scenarios that are not able to deal with UMM's complexity.

2.3.4. Alternative Services Choreography Languages

In the area of services choreographies, WS-CDL [223] and BPEL4Chor [31] are outstanding.

WS-CDL is a candidate recommendation of the W3C since 2005 and defines choreographies on the basis of WSDL. The definition of a WS-CDL choreography comprises static definitions and the actual specification of publicly visible message flow. The static definition comprises the definition of message types as either primitive XML Schema types or as WSDL `messages`. The participating roles of a choreography are defined as WSDL `portTypes` and some additional static declarations such as the message exchange relations between roles and the extraction of correlation information between participant instances can be defined.

The basic building blocks of the actual choreography definition are so-called *interactions* that define the one-way or request-response message exchanges between the roles defined beforehand. In order to specify the control flow between interactions, constructs like `parallel` for concurrent execution of interactions or `choice` for branching behavior can be used. Further, hierarchical decomposition is supported by specifying the execution of other choreographies using a so-called perform construct.

WS-CDL is an interaction-centric choreography language that is similar to ebBP in using a single- or two-message interaction as building block of potentially multi-party choreographies that may be comprise other choreographies. Contrary to ebBP, WS-CDL must be classified as implementation technology specific language due to its tight integration with WSDL. Furthermore, typical B2Bi QoS attributes cannot be attached to interactions.

BPEL4Chor already is briefly discussed in the introduction of this section. Conceptually, BPEL4Chor can be interpreted as a choreography layer that is added to BPEL and therefore supports the incorporation of existing BPEL definitions in a bottom-up manner [31]. A BPEL4Chor choreography consists of *participant behavior descriptions*, a *participant topology* and a *participant grounding*. The admissible language for defining participant behavior descriptions essentially is a sub-class of the BPEL Profile for Observable Behavior (cf. section 2.2) that disallows the definition of `partnerLink`, `portType`, and `operation` attributes for BPEL messaging activities like `onMessage` or `receive`. Like that, participant behavior descriptions do not technically depend on WSDL. Furthermore, participant behavior descriptions specify the sequence of messaging activities of one choreography role.

BPEL4Chor's participant topology defines the communication links between the participating choreography roles. Therefore, each participant behavior description is interpreted as a participant type and one or more participants are defined per type. In addition, so-called messageLinks define which kind of messaging activities are allowed to be performed between particular participants. Note, however, that this does not include the sequence of messaging activities that is implied by the participant behavior descriptions.

Finally, BPEL4Chor's participant grounding is used to bind messageLinks to concrete WSDL `operations` and `portTypes`. As this grounding is not unique, BPEL4Chor facilitates reuse of choreography definitions.

As the participant behavior descriptions are defined independently of each other and then connected by means of a participant topology, BPEL4Chor is classified as interconnection-centric choreography language. Interconnection-centric choreography languages frequently provide more flexible means for control flow definition because the behavior of the interacting roles can be specified independently (which is an implicit requirement for some service interaction patterns [11]). However, those languages do not lend themselves very well to hierarchical decomposition. Furthermore, BPEL4Chor must be classified as implementation technology specific choreography language. Although a BPEL4Chor choreography does not technically depend on WSDL, its underlying interaction paradigm relies on WSDL. Moreover, BPEL can be considered as implementation technology as well. Typical properties of B2Bi choreographies such as B2Bi QoS attributes or support for business document libraries are not available in BPEL4Chor.

2.3.5. Alternative Conceptual Choreography Languages

The largest set of alternative choreography languages is conceptual in nature and neither offers the typical B2Bi concepts of B2Bi choreographies nor the Web Services specific attributes of services choreographies.

BPMN *"Collaborations"* [150, section 9] represent the second choreography language offered by the BPMN standard. Contrary to BPMN choreographies, BPMN collaborations are interconnection centric and define the publicly visible message exchanges between entities by first describing the message exchange behavior of the individual participants (*pools* in BPMN terminology) and then defining so-called *message flows* between the communication activities of the participants. The rich set of constructs for defining BPMN *processes* [150, section 10] is available for the definition of participant behaviors. However, defining the participant behavior as a BPMN process is optional [150, section 9.5] and the message flows between participants may be attached to either activities within the participants' pools or to the pools directly (in case the behavior is hidden). Obviously, hiding the behavior of all participants makes little sense as the control flow definition of BPMN collaborations relies on participant behavior. Yet, hiding the behavior of all participants except for one focal participant allows for the definition of public orchestrations that then can be complemented with the behaviors of other participants. Another interesting functionality of BPMN collaborations is the integration with BPMN choreographies, i.e., the message flows between collaboration participants may be attached to the choreography activities of BPMN choreographies. While integrating both choreography styles of BPMN in one single diagram may be superfluous for the purpose of system specification, the combination offers a valuable feature for validating BPMN collaboration definitions against BPMN choreography definitions and vice versa.

Note that BPMN collaborations are fundamentally different from ebBP BCs. Whereas BPMN collaborations provide an interconnection-centric view on choreographies, ebBP defines an interaction-centric view. Similarly to BPMN *choreographies*, BPMN collaborations do not offer B2Bi specific concepts and can be specified independently of implementation technology. Hence, BPMN collaborations are classified as conceptual choreographies although they offer a notation that targets business process definition and although communication activities may be bound to WSDL (without visual presentation).

Petri nets [70] are a natural candidate for the specification of B2Bi scenarios due to their convenient representation of concurrency by means of different places.

Interorganizational Workflow Nets (IOWF-Nets) [214] are one of the first Petri net dialects that have been dedicated to inter-organizational processes. The basic concept of an IOWF-Net is the distinction between places for representing message buffers and places for reprensenting local states. Furthermore, receive and send transitions are used to link message buffer places and local state places. A receive transition consumes tokens from a message buffer place and a local state place and produces a token in a local state place. Conversely, a send transition just consumes a token from a local state place and produces a token in a message buffer place and another local state place. An IOWF-Net choreography then can be described by arranging the local states and communication transitions according to the participants (comparable to BPMN pools without boundaries) and placing the message buffers in between. Thus the message buffers link the corresponding send and receive transitions of the participants. The attractiveness of this model is that it is relatively easy to produce participant projections by interpreting send and receive transitions as interfaces for external communication and refining the local state places and transitions of one participant with private (i.e. invisible to other participants) places and transitions. Furthermore, the rich set of Petri net theory, algorithms and tools can be used to validate IOWF-Nets and its derivatives. Note that the work in [214] can be considered to be representative for a series of other Petri net based models for interorganizational systems such as the work in [35] that extends IOWF-Nets with refined notions of participant behavior.

So-called Interaction Petri nets [33] represent a fundamentally different way of applying Petri nets to B2Bi scenarios. Petri net transitions are not interpreted as local send or receive events, but as combined send and receive events of two interacting parties. In the terminology used in [33], transitions are denoted as *interactions* that consist of two roles and a message type. Interaction petri net places represent the overall progress of partner interactions (similar to concepts developed for the work at hand [160,182,186]) and interactions consume a token of a place and produce a token in another place. For example, assume that some place represents the agreement between two partners about the purchase of some product. Then an interaction that exchanges a delivery notification could be used to reach another place that represents the delivery of the product and yet another interaction for exchanging a payment advice could be used to reach a place that signifies successful payment. A choreography modeled as an Interaction Petri net correspondingly consists of a series of interactions that connect global choreography states.

IOWF-Nets as well as Interaction Petri nets, and therefore the numerous other Petri net approaches that rely on the same basic concepts, both offer abstract models for the specification of B2Bi scenarios. In consequence, B2Bi domain concepts or services technology specific artifacts are missing and neither a classification as B2Bi choreography nor as services choreography is justifiable. However, the models may be useful for analyzing dedicated B2Bi or services choreography languages. The availability of both, interconnection-style (IOWF-Nets) and interaction style (Interaction Petri nets) Petri net choreographies, promises that this is doable for most relevant choreography languages.

In [232], the Message Choreography Modeling Language (MCM) is described as choreography language that *"seamlessly complements existing models at SAP"* [232]. MCM defines choreographies between exactly two integration partners and tracks the progress of the choreography using *global states* (similar to [160,182,186] and Interaction Petri nets). Transitions between those global states represent message transmissions (or *interactions*) that are characterized by the message type exchanged, the message direction (between the two partners), and some guard that rules enabling of the transition. However, request-response interactions as available in ebBP, UMM, WS-CDL or BPEL4Chor are not available. A specialty of MCM is that the interpretation of choreography models requires the definition of so-called viewpoints, i.e., whether the choreography definition represents (at least) all possible send sequences of messages, (at least) all possible receive sequences of messages or (at least) all possible sequences of monitoring messages in transit. These viewpoints may deviate because asynchronous communication with message reordering is allowed for. The definition of different viewpoints reflects the purpose of MCM which is system analysis and testing rather than system specification (in the sense of providing an interaction contract that is to be implemented). As B2Bi specific or services specific concepts are missing, MCM clearly is an implementation-independent conceptual choreography language and the communication focus is interaction-centric.

Let's Dance [245] is a visual choreography specification language that claims *not* to be *"based on imperative programming constructs such as variable assignment, if-then-else and switch statements, sequence, and while loops"* [245]. The basic building block again is an interaction between two roles that models the exchange of a single message and an optional acknowledgment. Such an acknowledgment is used for notifying the sender about message delivery only and does not carry meaning. In particular, *"the protocol used for acknowledging is an implementation concern"* [245]. Therefore, message exchanges in [245] are essentially *one-way* and are not comparable to the *request-response* interactions of several choreography languages discussed above. The flow between interactions in Let's Dance is determined by the concepts of *dependencies*, *composition*, *loops*, and *guards*. Three different types of dependencies are available that enable a data flow-like execution description. A *precedes* dependency with interaction X as source and interaction Y as target expresses that Y only is admissible for execution if X has been performed before. Conversely, the *inhibits* dependency expresses that Y may not be performed any more once X has been performed. Finally, *weak-precedes* expresses that Y may be performed if either X has been performed or if X has been inhibited. Defining *precedes* and *weak-precedes* dependencies can be interpreted as means for passing on control flow between interactions. As these relationships may be defined between virtually arbitrary interactions (composition boundaries may not be crossed), Let's Dance can be said to support graph-like definition of choreographies. However, note that *precedes* dependencies are different from the transitions typically available in graphs. While a cycle of transitions is a loop, a cycle of *precedes* is a dead-lock. Composition of interactions may be performed hierarchically by simply delineating the corresponding interactions with a rectangle. Interactions within such a *composite interaction* that are not connected via dependencies are performed in parallel. In addition, *guards* can be used to restrict the eligibility of interactions for execution and three different types of looping behavior can be defined. While the *forEach* loop type may be used to iterate over sets of participants or variables, the *while* and *repeat until* loop types may be used to implement top-controlled or bottom-controlled loops using some variable-based condition expressions. Guards and loop constructs are attached to the boundaries of basic or composite interactions. As composite interactions may not overlap, this corresponds to a block-structured way of defining control flow. In summary, the control flow definition of Let's Dance indeed is different from the above choreography languages regarding the use of dependencies. However, the control of loops and the usage of variables is not unique and not really free of imperative programming constructs as claimed by the authors.

Although Let's Dance is designed such that the largest part of *service interaction patterns* [11] is supported, it is not classified as a services choreography because services technology constructs are missing. Similarly, B2Bi specific constructs as offered by ebBP, UMM or BCL are not available.

The Blindingly Simple Protocol Language (BSPL) [201] is a textual format for declaratively specifying interaction protocols between two or more roles that is fundamentally different from the other approaches discussed above in not using explicit constructs for representing control flow concepts such as repetition, parallel composition or branching. Instead, the sequencing of message exchanges is based on the availability of message parameters. Therefore, the definition of a message exchange

between two choreography roles includes a series of parameters that represent the information items on which the dependencies between messages rely. For example, the exchange of a *purchase* message sent from a *buyer* role to a *seller* role may require parameters for a *product* and a *price* that must be produced by a *quote* message from the *seller* to the *buyer* beforehand. In order to make dependencies clear, parameters are adorned with *in*, *out* or *nil* labels. The label *in* attached to a parameter means that the sender of the corresponding message must be aware of the parameter value beforehand whereas the *out* label means that the sender produces the value. The label *nil* means that the corresponding parameter is not bound yet. So, if the parameters *product* and *price* would be adorned *in* in the *purchase* message above and *out* in the *quote* message then the exchange of the *quote* message necessarily would have to precede the *purchase* message. The *nil* adornment is particularly useful for allowing different choreography roles to set the value of a parameter. So, if a second *quote* message was defined with the *price* parameter adorned with *nil* then the *buyer* could produce a value for the *price* parameter. Of course, a second definition of the *purchase* message with an *out* adornment for *price* would be necessary for this as well. Note that the sequence of message exchange definitions in BSPL is irrelevant because the message dependencies are determined via parameter adornments.
Beyond the definition of message exchanges, the only additional concept of BSPL is protocol composition. For composing protocols, the roles of a superordinate BSPL protocol are mapped to the roles of a subordinate BSPL protocol and the parameters consumed and produced by the subordinate protocol must be given. So, BSPL is indeed *blindingly simply* in terms of the number of modeling constructs used. Yet, it may be *hard* to specify advanced control flow scenarios based on *in*-, *out*-, and *nil*-adornments of messages only.
BSPL is classified as conceptual choreography language that particularly focuses on capturing information dependencies between messages. Although the explicit representation of message parameters in the sense of information items that determine control flow make BSPL particularly interesting for business interactions, the lack of B2Bi concepts such as QoS attributes forbids the classification as B2Bi choreography. BSPL is deliberately implementation technology agnostic and centered around the interactions of the partners.

Beyond BPMN collaborations, IOWF-Nets, Interaction Petri Nets, MCM, Let's Dance and BSPL, a myriad of languages can be interpreted to support choreography aspects. Communication between system components is a core computer science aspect and hence a wide variety of languages aims at or is adapted to this issue. For example, process algebra languages such as the *π-calculus* [114, 115], *Communicating Sequential Processes* [58] or *Calculus of Communicating Systems* [113] provide sound formalizations of interacting systems and therefore may lend themselves well for analyzing cross-organizational processes. The adequacy of process algebras for capturing cross-organizational message exchange relationships is consistently reflected in the development of more recent languages such as the *piX model* [216], the *Lightweight Coordination Calculus* [18, 167] or the *Multiagent Protocols* language [8] for the purpose of formalizing choreographies. However, process algebras do not support the communication function of choreography models very well. Similarly, UML *activity diagrams* or *interaction diagrams* [149] may be adapted to representing cross-organizational processes, but their actual purpose rather is system specification. So, this section just gives an overview of those languages that come closest to ebBP in terms of practical relevance for specifying B2Bi choreographies.

2.4. ebXML

The initial version of ebBP was developed in the course of the ebXML project as part of a comprehensive B2Bi framework. The *Core Components Technical Specification (CCTS)* [209], the *Collaboration-Protocol Profile and Agreement (CPPA)* [128], the *ebXML Registry Information Model (ebRIM)* [131], *ebXML Registry Services and Protocol (ebRS)* [132], and the *ebXML Messaging Services (ebMS)* [129, 136] are available for addressing core B2Bi issues not covered by ebBP.

Figure 2.11 visualizes the interplay of ebBP with these related standards. The box labeled *Business Process Definition* in the upper-left corner of figure 2.11 represents the actual application area of ebBP. ebBP specifies the exchange of business documents using BTs and BCs, but does not cover the definition of business documents. These are assumed to be ready for import from business document libraries and CCTS provides an implementation-neutral standard for defining so-called *core components* as building blocks of business documents with unambiguous semantics (upper part of figure 2.11). ebBP defines constructs for specifying different technical formats of business documents and allows for an extensible set of expression languages for defining choreography control flow based on the contents of the exchanged business documents. While a basic set of constructs is needed for capturing the role of business documents in B2Bi choreographies, the details of defining business documents are irrelevant for choreography specification. Hence, CCTS is not further investigated in the context of this work.

According to the ebXML deliverables, process definitions (or rather B2Bi choreography definitions), business document definitions and core components are assumed to be stored in some repository with an attached registry (center part of figure 2.11) so that business partners can easily query and retrieve artifacts. ebRIM and ebRS are available as standards for such repository and registry services. As this work is not concerned with the retrieval of choreography or business document definitions (cf. section 1.1), ebRIM and ebRS are not further discussed.

CPPA and ebMS are available for covering the more technical aspects of business document exchanges. The CPPA standard specifies the definition of so-called Collaboration-Protocol Profiles (CPPs) that describe the message exchange capabilities of one partner and of so-called Collaboration-Protocol Agreements (CPAs) that describe the message exchange capabilities that two particular partners have agreed upon (center part of figure 2.11). CPPs and CPAs basically leverage the same set of constructs, but obviously differ in the multiplicities of the used constructs. In order

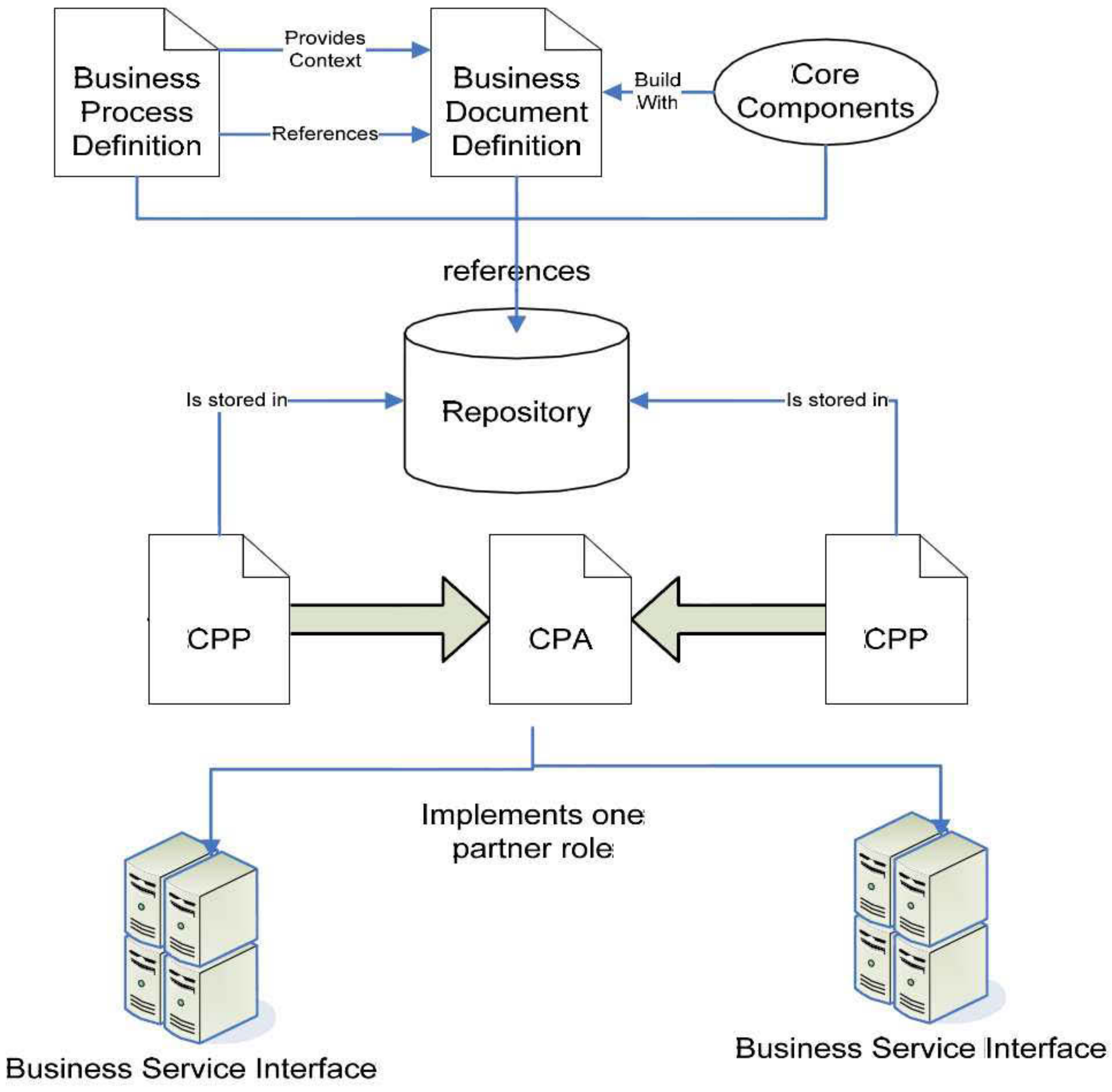

Business Process Definition
Provides Context
References
Business Document Definition
Build With
Core Components
references
Repository
Is stored in
Is stored in
CPP
CPA
CPP
Implements one partner role
Business Service Interface
Business Service Interface

Figure 2.11.: ebBP and Complementary ebXML Standards, taken from [134]

to describe the message exchange capabilities of an integration partner, the CPPA standard refers to ebBP concepts. The capabilities of a particular integration partner are described in a `PartyInfo` element that relates the partner to one or more of the roles of (typically) ebBP BCs. Then, `CanSend` and `CanReceive` elements are used to describe how the business documents or business signals defined within a BC are supposed to be exchanged. For identifying the respective BC declarations, a so-called `ActionBinding` is used in CPPA. Note that CPPA does not explicitly require the use of ebBP for defining a *collaboration context*, but the technical concepts are tightly integrated which is formulated in the CPPA standard as follows:

> *"A Party can describe the Business Collaboration using any desired alternative to the ebXML Business Process Specification Schema. When*

> *an alternative Business-Collaboration description is used, the Parties to a CPA MUST agree on how to interpret the Business-Collaboration description and how to interpret the elements in the CPA that reference information in the Business-Collaboration description."* [128, section 8.4.4]

The actual technical configuration of the message exchanges within `CanSend` and `CanReceive` elements is given by so-called `DeliveryChannels`. These, in turn, consist of a `Transport` element for specifying the transport protocol to be used (such as HTTP or FTP) and a `DocExchange` element for configuring the messaging agents that perform business document/signal exchanges on top of the transport protocols. The parameters that are available for a `DocExchange` element are defined using so-called `SenderBinding` and `ReceiverBinding` elements. Although CPPA allows for arbitrary types of such bindings for covering any messaging protocol, only an `ebXMLSenderBinding` and `ebXMLReceiverBinding` are provided that are tied to ebMS: *"The ebXMLSenderBinding element describes properties related to sending messages with the ebXML Message Service[ebMS]"* [128, section 8.4.40].

ebMS defines the messaging level protocol for exchanging business documents and business signals on top of SOAP. This is in line with the SOAP specification that does not define a complete messaging protocol itself but rather a framework for defining those. Several Web Services technologies such as WS-ReliableMessaging [143] or WS-Security [135] are suggested to be used for implementing QoS features. However, note that service descriptions based on WSDL are not available that make up the actual power of Web Services technology (cf. section 2.1). Although `WS-SecurityPolicy` and `WS Reliable Messaging Policy Assertion` are included in the *normative references* section of ebMS version 3, the details of using these standards are not given. So, ebMS is not a true Web Services technology, but rather a SOAP-based messaging protocol for B2Bi.

ebMS version 2 [129] significantly relies on CPPA concepts which is reflected in the standard itself as follows:

> *"As regards the MSH [ebMS Message Service Handler], the information composing a CPP/CPA must be available to support normal operation. However, the method used by a specific implementation of the MSH does not mandate the existence of a discrete instance of a CPA"* [129, section 1.2.3]

The dependency on CPPA becomes manifest in several required message headers such as `CPAId` or `PartyId` although ebMS allows for filling in values that are taken from other sources than a CPA. In ebMS version 3 [136], the dependency on CPPA is further relaxed although some elements such as `PartyInfo` or `CollaborationInfo` still are defined as required in message headers.

Summarizing the references defined in CPPA and ebMS to other ebXML technologies makes clear that significant conceptual dependencies between the ebXML layers exist. Although ebXML is frequently said to allow for easy integration of non-ebXML technologies, the tight dependencies between ebXML layers suggest that this requires significant effort. In this work, ebBP as one of the most abstract ebXML technologies will be used for defining choreographies and non-ebXML technologies will be used for implementation. A discussion on why ebMS and CPPA in particular have not been used for specifying and implementing the technical message exchanges of B2Bi choreographies is provided in the introduction of chapter 5.

3. Requirements and Design Choices

This section presents the goals, method and results of the B2Bi requirements analysis reported on in [184] and discusses how those requirements affect this work. In fact, the results of the study help in narrowing down the types of B2Bi scenarios to be supported. The intent of the requirements study [184] is

identifying and classifying requirements for the analysis, design, development and maintenance of B2Bi information systems.

Different types of requirements sources are analyzed for enhancing both the validity and the completeness of the requirements set. Firstly, the functionality of dedicated integration standards such as ebBP, UMM or RosettaNet's RNIF is investigated [123, 128, 131, 132, 134, 136, 170, 208]. Secondly, reference architectures for B2Bi and BPM are examined [13, 87, 199]. Finally, scientific literature with focus on SCM, BPM, B2Bi, enterprise integration, and language assessment is investigated [3, 9, 34, 47, 88, 89, 100, 101, 125, 126, 148, 161, 162, 176, 187, 193, 213, 234, 243]. The study results in 78 B2Bi requirements. This number calls for additional classification in order to be beneficial. 7 core B2Bi challenges such as *communication among unequal personnel*, *management of complex associations* or *homogenization of computing resources* are identified that are typical for B2Bi scenarios. Requirements are then classified according to their utility for overcoming those challenges. As those challenges vary with different types of B2Bi systems (cf. section 1.1.3), such a classification is helpful for selecting requirements to be addressed for a particular B2Bi project. In addition, requirements are classified according to the abstraction layers of the B2Bi schema of figure 1.3 (p. 12). Each requirement was evaluated with respect to the abstraction layer on which it is supposed to be addressed. As the abstraction layers of the schema tightly correlate with development phases of B2Bi projects, such a classification helps in identifying the point in time when a requirement should be addressed.

In the next two sections, the approach for performing the requirements study as well as the results of the study are presented.

3.1. Approach of the Requirements Study

The approach taken for the study is aligned with three main goals. Firstly, a list of requirements for the *analysis, design, development and maintenance* of B2Bi information systems should be developed that helps researchers in identifying research opportunities and assessing completeness of research plans as well as practitioners in evaluating B2Bi projects and tool sets. Secondly, this list should be comprehensive. Thirdly, the list should be manageable.

The first goal pays tribute to the influence of B2Bi on today's business world. Moreover, it makes clear that the focus of this work is technical. The managerial and strategic aspects of SCM are not considered as stated in section 1.1.1. Moreover, *project management and organizational issues* during performing B2Bi projects are disregarded. Project management typically comprises tasks like risk management, human resource allocation or project scheduling, while organizational issues cover aspects like cultural fit, implementing organizational change, level of support for IT projects or the analysis of organizational capabilities. Exemplary publications that are dedicated to these issues are [88] investigating EAI success factors, [119] defining a capability assessment framework for the adoption of B2Bi Systems or [90] presenting decisive components of a SCM framework. The second goal, comprehensiveness, leads to the selection of different requirements sources as described below. Moreover, as the scope of this work comprises the design and development of information systems, generic requirements for models and implementations apply. Examples of such requirements are *coupling*, *cohesion*, *abstraction* or *reuse*. However, this work does not even attempt to define a comprehensive requirements list for arbitrary models. Instead, a comprehensive list of B2Bi requirements that can be justified by relevant B2Bi requirements sources is sought for. The third goal, manageability, contradicts comprehensiveness to some extent and leads to the definition of aggregated requirements. To further enhance manageability, the relation between requirements and B2Bi challenges as well as the abstraction layers of the B2Bi schema (cf. figure 1.3; [187]) are also examined in more detail.

Three different types of sources are selected for deducing B2Bi requirements, namely dedicated B2Bi standards, B2Bi/Business Process reference architectures and scientific literature. Thus, different views on the same subject of investigation can be consolidated which leads to a more comprehensive treatment of the topic. The rationale behind choosing B2Bi standards is that requirements are driving implementation artifacts. B2Bi standards are, in effect, used for implementing B2B collaborations and thus the investigation of the functionality of these standards leads to B2Bi requirements. Furthermore, B2Bi standards are usually developed by domain experts and, therefore, this type of requirements source also enables access to expert knowledge.
B2Bi/Business Process reference models/architectures are the second type of requirements source selected. Reference models/architectures describe best practice knowledge on which components to select for representing the subject under consideration and on how to relate these components. The *qualities* of such reference models/architectures may be used for evaluation purposes (cf. [89]) and define requirements at the same time. Hence, examining the qualities of the components and the relationships among these reveals the intended requirements.
The third type of requirements sources is scientific literature. The following rule is used for eliciting B2Bi requirements from literature: Either the paper contains

an explicitly defined requirement or the authors describe some functionality/property/method/concept/tool as particularly useful or not useful. The sheer description of a functionality/property/method/concept/tool without valuation is not sufficient for deducing requirements. For the work at hand, different categories of papers have been searched for. The first category comprises surveys and reviews about requirements for B2Bi. This category is scarcely occupied. Furthermore, B2Bi and BPM surveys are considered. Thereby, the claim is made that BPM is tightly related to B2Bi because business collaborations can be interpreted as enterprise-spanning business processes. Third, papers that define requirements for some sub-category of B2Bi and BPM are considered to be relevant. This category comprises topics like Web Services compositions as technique for implementing B2Bi or EAI which is an important BPM task. Finally, drivers and obstacles for adopting B2Bi are selected as a category of focal publications.
The literature search itself has been performed using scientific search engines like IEEE Xplore[1], ACM Portal[2] or Google Scholar[3] and by systematically searching relevant journals, conference proceedings and institutional homepages.

For eliciting requirements each reference, i.e., B2Bi standard, reference model/architecture or publication has first been evaluated in isolation and all requirements found have been written down separately. This unconsolidated list comprised several hundred requirements which is caused by identical requirements defined in different references and by very detailed requirements lists that can be deduced from some sources such as [126] or [134]. Therefore, related requirements have been aggregated, for example 11 consistency requirements identified in [187] have been merged to simply *consistency*, and checked for semantic equivalence by investigating the related references again. Eventually, 78 aggregated B2Bi requirements have been identified. For enabling traceability of the origin of requirements, the related sources have been associated in tables A.1 and A.2 (pages 290, 294). This matrix not only enhances traceability but also gives an impression of how frequently a requirement is identified. A true statistical analysis is not possible because the sample of requirements sources is necessarily influenced by the background of the researchers and therefore not random.

The classification of requirements according to B2Bi challenges and B2Bi schema abstraction layers resulted from collaboration with my fellow researchers Christian Wilms and Guido Wirtz in a two-step process. At first, all researchers, who all are engaged in distributed systems, workflow and business process modeling, classified the requirements individually. Then, the results of individual classification have been compared and merged in several discussion sessions which lead to matrices A.3 (page 297) and A.4 (page 300). Please see appendix A for more details on how the individual classifications were created and for instructions on how to read the tables.

[1]`http://ieeexplore.ieee.org/Xplore/`, last access: 12/20/2011
[2]`http://portal.acm.org`, last access: 12/20/2011
[3]`http://scholar.google.com/`, last access: 12/20/2011

3.2. Results of the Requirements Study and Design Choices

The surveyed standards, reference architectures and literature result in 78 B2Bi requirements. Instead of discussing all 78 B2Bi requirements identified, the taxonomy of 7 core B2Bi challenges that can be overcome by addressing those requirements is presented. Note that not all requirements are relevant for the scope of this thesis (cf. section 1.1) so that a restricted set of relevant requirements is presented subsequently. Finally, the support of requirements by the core artifacts of this work, i.e., ebBP-Reg, ebBP-ST and SeqMP is discussed.

Table 3.1 presents four original and three derived challenges that can been identified for B2Bi systems. The four original challenges represent the typical setting of B2Bi scenarios and cover *communication among unequal personnel*, *agreement*, *management of complex associations* and *homogenization of computing resources*. Thinking about a simple *"quote and order"* scenario with personnel from two different enterprises having to agree upon what data to exchange and when to perform business tasks depending on the message exchanges while considering that a similar process may have to be performed with different business partners using various communication technologies, these challenges become obvious. But, these challenges also come in different flavors. Agreement does not only concern the specification of particular message formats and a business process that reacts to messaging events. It is also a question of legal implications of message exchanges. Similarly, homogenization of computing resources not only concerns differing computing platforms of integration partners and interfacing with legacy systems. It is also about catering for the characteristic issues of distributed systems like partial failure and lost/duplicated messages.
The three derived challenges *comprehensibility*, *feasibility* and *changeability* are more generic in nature and therefore also apply to other systems than B2Bi systems. Nonetheless, they have been included as they are particularly challenging in the B2Bi context. Comprehensibility is especially hard in the face of communication among unequal personnel, management of complex associations and homogenization of computing resources, but it is indispensable for achieving agreement. Similarly, feasibility as a second precondition for agreement needs special attention when considering communication among unequal personnel and homogenization of computing resources. Finally, the importance of changeability can be derived from management of complex associations and homogenization of computing resources.
There may be different valid systematizations of B2Bi challenges. The systematization presented has been developed during the requirements study of [184] on the basis of B2Bi experience of the authors. However, there is an empirical indicator for the validity of the challenges scheme as 76 out of 78 B2Bi requirements can be associated with a B2Bi challenge the requirement particularly helps solving in.

Table 3.1 shows the challenges together with the respective variants and relations between original and derived challenges.

Index	Type	Challenge	Variants	derived from	essential for
1	original	Communication among unequal personnel	- Across enterprises - Business analyst to IT expert		
2	original	Agreement	- Legal - Business - Technical		
3	original	Management of complex associations	- Dynamics with respect to *well-known* partners and partner links - Dynamic binding to *unknown* partners - Multi-party collaborations		
4	original	Homogenization of computing resources	- Legacy systems - Platform heterogeneity - Distributed computing		
5	derived	Comprehensibility		1,3,4	2
6	derived	Feasibility	- Business appropriateness - System appropriateness	1,4	2
7	derived	Changeability	- Extensibility - Replaceability	3,4	

Table 3.1.: Overview of B2Bi Challenges

Considering the large variety of challenges and requirements, it is naive to think that all requirements can be met by one single approach. The number of requirements can be narrowed down by focusing on the research question of this thesis (cf. section 1.2) and by focusing on the type of B2Bi systems that define the scope of this work (cf. section 1.1.3). The associations between requirements and core B2Bi challenges as well as between requirements and B2Bi abstraction layers then help in selecting relevant requirements.

Considering that the research question targets the semantic gap between *business process models* and *private orchestrations* reveals that requirements that are particularly important for the *business model abstraction layer* or the *runtime abstraction* *layer* may be of minor importance for the work at hand. Similarly, the core challenge *management of complex associations* is less important when focusing on the *extended enterprise* type of B2Bi because interactions with completely unknown partners are rare and hence requirements that particularly help in overcoming this challenge are candidates for being neglected. Likewise, focusing on *decentralized execution of choreographies* implies that requirements that help in tackling the *agreement* challenge are particularly important and the focus on *straight through processing* implies particular importance of requirements that help in overcoming the *communication among unequal personnel*, *agreement* or *feasibility* challenges.

The derivation of requirements along those considerations results in the following requirements list that should be met by this work.

1) **Usage of standards.** Standard choreography and orchestration languages are to be used.

2) **Language technical actor appropriateness.** Choreography and orchestration models should be amenable to automatic processing which has several implications:
    a) Machine-processable format. The syntax of models is to be precisely defined, e.g., using XML Schema technology.
    b) Clear semantics. The meaning of language constructs must be precisely defined.
    c) No deviations from standards. The use of standard tools should not be hindered by deviations from choreography or orchestration standards.

3) **Support for business documents.** The import for existing business document definitions as defined by business document libraries like RosettaNet or Odette is to be supported. For an extensive discussion of business document standards, see [95].

4) **Language domain appropriateness.** Choreography and orchestration languages should reflect the characteristics of the B2Bi domain. This includes:
    a) Support for business transactions. The concept of business transaction should be used to abstractly define alignment of the integration partners' IT systems at the choreography level. At the orchestration level, the details of interaction are to be specified.
    b) B2Bi Quality-of-Service. B2Bi-related QoS features like security or reliable messaging are to be supported.
    c) Data oriented process definition. It should be possible to describe the routing logic of the sequences of admissible business transactions using the data that has been exchanged.
    d) Support for roles. Roles should be supported to allow for the abstract definition of tasks of integration partners that then can be mapped to concrete partner instances.

e) State-based modeling. Changing the state of the integration partners' IT systems is the goal of B2Bi processes and state typically influences the applicability of business transactions. State therefore should be adequately represented in choreography and orchestration models.

f) Interfacing with business applications/communication interface. B2Bi projects have to consider the integration with business applications and therefore the corresponding interfaces should be defined.

5) **Language comprehensibility appropriateness.** Choreography and orchestration languages should be easy to understand.

6) **Technology independence of process model.** At the choreography level, B2Bi processes should be defined in a technology-agnostic way in order to allow for different messaging technologies like Web Services, ebMS [136] or AS2 [116]. This comes in two flavors:

   a) Support for multiple communication technologies in new processes.

   b) Support for multiple communication technologies when reusing existing interactions.

7) **Control flow definition.** Reasonable expressiveness for control flow definition is needed.

   a) Hierarchical decomposition. Composing/decomposing complex interactions should be possible.

   b) Support for multi-party collaborations. The definition of interactions between more than two integration partners should be supported.

   c) Control flow/interaction patterns. Control flow/interaction patterns like those defined in [174, 211] or [11] should be supported.

8) **Error handling.** The handling of errors in performing B2Bi orchestrations must be defined.

9) **Extensibility.** Business transactions should be extensible in order to allow for new types of interactions.

10) **Formalization.** Formalization of choreography and orchestration models provides the foundation for automated translation, validation, simulation or semantic constraint management.

   a) Formalization of input models. The classes of valid choreography models have to be defined formally.

   b) Clear execution semantics. The semantics of executing choreographies should be defined formally.

The basic concept for meeting these requirements is to leverage a B2Bi choreography format as contractual agreement between integration partners that then can be used for deriving orchestration-based implementations or for analysis of the interactions. However, the intent of this work is not to come up with completely new languages to meet these requirements. Instead, existing standard languages should be reused *as is* and complemented where necessary. For this purpose, ebBP as B2Bi choreography language and BPEL as orchestration language are selected.

ebBP better suits the requirements defined above (index in parantheses) than the choreography languages identified in section 2.3. ebBP allows for the import of existing business document definitions (3), the messaging technology agnostic definition of business document exchanges using the concept of business transactions (4a, 6a/b), routing expressions defined on business documents (4c) as well as the specification of B2Bi relevant QoS parameters (4b). ebBP BusinessCollaborations can be used to choreograph ebBP BusinessTransactions (and other BusinessCollaborations) using control flow constructs like decisions, forks and joins (7a, partly 7c). Multiple roles (7b) can be defined at the level of BusinessCollaborations that then are mapped to the (exactly two) roles of BusinessTransactions (4d). As XML-based B2Bi standard, ebBP naturally supports requirements 1 and 2a.

At the orchestration level, the decision of using BPEL as orchestration language is driven by the choice of Web Services as most important messaging technology. At the business transaction level, ebMS and AS2 are considered as relevant alternative implementation technologies (6a/b), but interactions for implementing control flow between business transaction executions as well as integration with business applications is assumed to be implemented using Web Services due to its interoperability benefits. Business transaction implementations using Web Services and BPEL have been researched in [172, 181, 192] and therefore the standards-based realization of B2Bi-relevant QoS attributes as well as sufficient means for error handling can be assumed to be realistic (requirements 1, 2a, 3, 4a/b/c/d, 8).

The discussion so far shows that important B2Bi requirements can be addressed by simply selecting the ebBP-BPEL tool chain. The real challenge rather is defining a precise semantics for ebBP which has neither been formalized nor unambiguously described beforehand (2b, 10a/b), defining a suitable integration architecture for performing B2Bi orchestrations (4f), allowing for more than one messaging technology in the implementation of a single business collaboration (6a/b), and weighing up comprehensibility (5) and standard compliance (2c) against state-based modeling (4e), support for control flow features (7a/b/c) and extensibility (9). For example, multi-party collaborations are harder to design and understand than binary collaborations while explicitly representing state in B2Bi collaborations fosters comprehensibility and impairs ebBP standard compliance. This work therefore defines different types of ebBP modeling for satisfying different integration scenarios. In addition, visualization of these modeling styles leveraging BPMN choreography notation is investigated because there are natural limits to comprehensibility and communication without appropriate visualization. Below, the different styles of ebBP modeling are associated with the requirements they fulfill.

Shared-state based ebBP modeling (ebBP-ST) targets comprehensibility and state-based modeling by explicitly modeling so-called *shared states* and limiting admissible ebBP models to business collaborations with exactly two roles and no support for parallel task executions, advanced interaction patterns or hierarchical decomposition. This leads to a state-machine like choreography definition as depicted

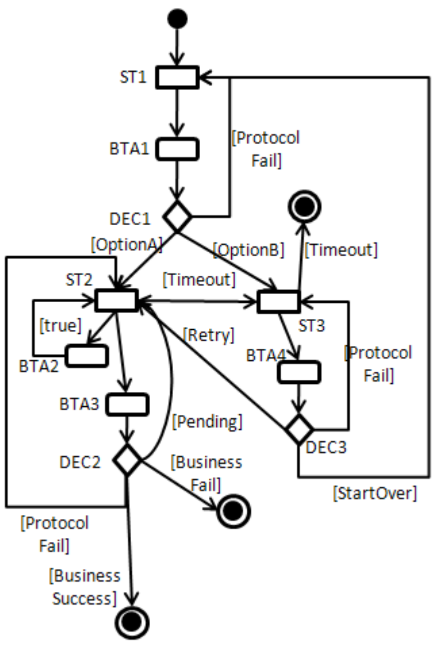


Figure 3.1.: Valid ebBP-ST Model

in figure 3.1 which can smoothly be translated into BPEL orchestrations[4]. Both partners of an ebBP-ST model concertedly leave and enter shared states (rectangles labeled `ST`<X> in figure 3.1) by performing business transactions (rounded rectangles labeled `BTA`<X> in figure 3.1) that consistently align state between integration partners. Every BTA is followed by a decision node (diamonds labeled `DEC`<X>) that makes routing between BTAs explicit. In figure 3.1, arrows visualize transitions where transitions that emerge from a shared state either are triggered by the execution of a BTA or a distributed timeout (denoted [Timeout]). Transitions that emerge from BTAs are triggered upon completion of the BTA and either directly link back to the shared state the BTA was triggered from or link to a decision that evaluates the BTA outcome. Finally, transitions that emerge from decision nodes immediately are triggered and represent the different outcomes of a BTA (captured as boolean guards in brackets).

[4]Figure 3.1 is an ad-hoc visualization of ebBP-ST chosen for reasons of compactness

ebBP-ST fosters comprehensibility in a twofold way: Shared states allow for explicitly reasoning about the applicability of performing business transactions and for reasoning about the results of business transaction executions while disallowing parallelism allows for almost arbitrary graph structures. Note that this class of ebBP models still is sufficient for capturing a large set of real-world B2Bi scenarios (cf. [166]). ebBP-ST modeling requires the introduction of shared states to the ebBP standard. Although shared states may be represented in an ebBP-compliant way, modeling efficiency calls for an extension (cf. [160]). In so far, fulfillment of requirement 2c is limited to some extent.

Regular ebBP modeling (ebBP-Reg) is CHORCH's second proposed ebBP modeling flavor that is more expressive in terms of control flow features than ebBP-ST. Most notably, parallel task execution and hierarchical decomposition are allowed for. In such an environment, the concept of shared states cannot easily be supported without substantial modifications of the ebBP standard and therefore shared states are dropped. In so far, ebBP-Reg can be considered to be a variation of ebBP-ST. Most notably, the underlying modeling paradigm again is state-machine based. In order to support this paradigm, ebBP-Reg imposes slight restrictions on standard ebBP for solving semantics issues and ensuring translatability into BPEL. This concerns clarifications of how to model the necessary information for routing after having performed a business collaboration within a different collaboration and the modeling of parallel structures (and only of parallel structures) as defined in [76]. These restrictions do not contradict the guidelines of the ebBP standard and therefore can be considered to be standard compliant.

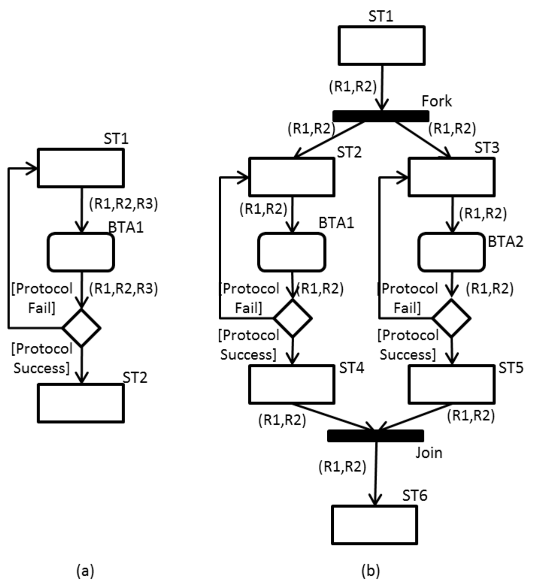


Figure 3.2.: ebBP+ Business Transaction and State Split

Extended ebBP modeling (ebBP+) is CHORCH's third proposed ebBP modeling flavor and combines advanced control flow features with the concept of shared states at the cost of dropping standards compliance. ebBP+ applies a Petri-net like modeling technique that allows multiple collaboration partners to be in multiple and/or different shared states (rectangles in figure 3.2). For controlling state alignment, the integration partners that must be in a shared state before performing a BTA (rounded rectangles in figure 3.2) are explicitly modeled by adding the corresponding role names to the incoming transition of business transactions. For example, figure 3.2 (a) uses the text label (R1,R2,R3) to require ebBP partner roles `R1`, `R2` and `R3` to be in shared state `ST1` before `BTA1` can be triggered. Similarly, the result of BTAs is also tied to participating integration partners by adding role names to outgoing transitions. By expressing preconditions and results of business transaction executions in terms of shared states, it is also possible to express interaction patterns as defined in [11] or new transaction types (requirements 7c, 9). Furthermore, parallel structures are supported by using Fork/Join nodes (visualized as black horizontal bars labeled `Fork/Join` in figure 3.2 (b)) to split up/combine shared states into/from sub-states that are modified by different BTAs (figure 3.2 (b)). Incoming transitions of Fork/Join nodes immediately fire once the respective ebBP roles have entered the source states (denoted as text labels on the transitions). Outgoing transitions of Fork/Join nodes always immediately fire and associate the ebBP partner roles specified on the transitions with the target states.
Note that ebBP+ is not further researched in this work due to the insight that the vast majority of B2Bi processes are not multi-party choreographies with synchronization between all participating roles. When analyzing 100 scenarios of the publicly available *RosettaNet implementation guides* (for implementing B2Bi processes), the majority of interactions was discovered to be binary (84 scenarios), i.e., performed between exactly two integration partners. This is in line with academic research, e.g., [79, 232]. Yet, the sheer existence of SCM implies that business processes touch more than just two roles. The SeqMP choreography style below is designed to cater for this paradox.

Sequential Multi-Party Modeling (SeqMP) is both, an ebBP modeling style and an analysis framework. The underlying paradigm is the implementation of a business process as a sequence of binary (component) choreographies between changing integration partners. Thereby, very limited assumptions are made about component choreographies. The results of these component choreographies are assumed to be synchronized between the participating roles and results are assumed to be distinguishable from each other by a set of names. This could easily be provided by defining and labeling several end states for component choreographies. Expressions defined on result names of a component choreography are then used for determining the follow-on component choreography. In so far, routing between component choreographies is based on data, but there is no elaborate data model (requirement 4c). Component choreographies are not assumed to be processable in parallel as this may result in intricate synchronization problems within backend systems. Hence, SeqMP can be interpreted as a state-machine where each component choreography is a state and the transitions correspond to the switch between two subsequent component choreographies (requirements 4e and 5). Hierarchical decomposition is supported in the sense that binary choreographies are the components of multi-party choreographies (requirement 7a). However, SeqMP choreographies are not eligible as components of other choreographies because the results of multi-party choreographies cannot simply be assumed to be synchronized between the participating roles. This, in turn, may result in deviating states of each role at the multi-party level.

Requirement	ebBP-ST	ebBP-Reg	SeqMP	ebBP+[5]
1. Usage of standards	0	+	+	-
2 a. Machine-processable	+	+	+	+
2 b. Clear semantics	+	+	+	+
2 c. No standards extensions	0	+	+	-
3. Business documents	+	+	%	+
4 a. Business transactions	+	+	%	+
4 b. B2Bi QoS	+	+	%	+
4 c. Data orientation	+	+	0	+
4 d. Roles	+	+	+	+
4 e. State-based modeling	+	0	+	+
4 f. Interfacing with backend systems	+	+	%	+
5. Comprehensibility	+	0	+	-
6 a/b. Technology independence	+	+	+	+
7 a. Hierarchical decomposition	-	+	0	+
7 b. Multi-party collaborations	-	-	+	+
7 c. Control flow/interaction patterns	0	0	%	+
7 d.* Parallelism	-	+	-	+
8. Error Handling	+	+	0	+
9. Extensibility	-	-	+	+
10 a. Formalization of input models	+	+	+	+
10 b. Formal execution semantics	+	+	+	+

Table 3.2.: CHORCH's B2Bi Choreography Modeling Flavors

The focus of SeqMP modeling is not implementation, but analysis. The target of the analysis are so-called *synchronization deficits* of integration partners that may result from erroneous execution of component choreographies. Assume that

[5]Not covered in this work

some role C still expects to participate in some component choreography, but that a preceding component choreography between roles A and B fails due to some business disagreement. C may not be notified about this circumstance which constitutes a *synchronization deficit*. Depending on the structure of the multi-party choreography, this situation is both realistic and hard to analyze. The analysis framework of SeqMP allows for identifying synchronization deficits and for configuring the analysis algorithm in terms of how synchronization deficits are established and resolved. However, while the identification of synchronization deficits is supported, the actual handling of the deficits is not addressed (requirement 8).

Note that the atomic building block of a SeqMP choreography is a binary component choreography where ebBP-Reg or ebBP-ST are available as specification format. Therefore, lower level requirements such as business document/business transaction support, interfacing with backend systems or control flow/interaction pattern support are not applicable. SeqMP deliberately abstracts from these details and although an ebBP representation is available for ensuring machine-processability (requirement 2a), SeqMP is defined independently of ebBP. That is why SeqMP is extensible in the sense of allowing for different component choreography models as long as these fulfill the basic assumptions made above. In particular, the algorithms for analyzing SeqMP models are designed such that they work for multi-party component choreographies as well, provided that the results are synchronized.

Table 3.2 summarizes the support of B2Bi requirements by CHORCH's choreography modeling flavors. The commonalities are due to the inherent advantages of using ebBP and BPEL as B2Bi choreography/orchestration language and by the need for a suitable integration architecture as well as for support of multiple messaging technologies in a single business collaboration.

Having given an overview of how CHORCH's choreography styles address the identified B2Bi requirements, the next chapter will introduce the styles in detail.

4. Representing B2Bi Choreographies

The representation of B2Bi choreographies decisively depends on the integration scenario and on the underlying development process model. B2Bi choreography models may be used in an informal and imprecise way which serves as basis for discussion during implementation or as stringent specification that uniquely determines part of the implementation. In this work, the former type of choreography models is referred to as *cartography choreographies* whereas the latter is referred to as *strict choreographies.*

Cartography choreographies primarily help in supporting *comprehensibility* and *communication between different personnel* (cf. B2Bi challenges of table 3.1), in particular between the business experts of the interacting parties. These use cartography choreographies for identifying the types of business documents to be exchanged, the ideal flow of message exchange sequences and relevant communication roles. However, not all admissible message exchange sequences are determined. The exact control flow logic for handling technical or business errors may be left out and/or some part of control flow logic may just be captured in prose. Consistently, cartography choreographies do not have an exact operational semantics and intensive interaction between business experts and software engineers is needed for deriving the implementation. The advantage of using such choreographies is separation of business discussion from technical discussion so that the evaluation of business appropriateness does not interfere heavily with the evaluation of systems appropriateness (cf. table 3.1).

There are three major drawbacks of cartography choreography modeling. Firstly, lacking systems appropriateness may necessitate complete rework of choreography models. Software engineers may detect irresolvable control flow definition flaws or reveal to business experts that choreography models contain unintended behavior. Secondly, it is hard to tell in how far the implementation of B2Bi systems conforms to requirements. As cartography choreographies do not have an exact semantics, software engineers may be tempted to just make assumptions about intended behavior or simply misunderstand requirements. In both cases, the result is unintended behavior of the implementation. Even worse, the lack of semantics impedes the application of tools that help in detecting deviations of the implementations from the choreography specification. Thirdly, interoperability between BSIs is threatened by deviating interpretations of the choreography models by the software engineers of the interacting parties. To prevent this, extensive discussions are needed to align the BSI implementation of the interacting parties which may result in retriggering discussions between business experts and software engineers and in the end may require the respecification of the choreography models by the interacting business experts.

Strict choreographies are different from cartography choreographies in offering a precise operational semantics that uniquely determines the set of admissible choreography executions. Beyond *communication between business experts* and *comprehensibility*, strict choreographies thus also support *agreement* about the meaning of interaction specifications and *system appropriateness* as a *feasibility* aspect (cf. B2Bi challenges of table 3.1). For offering a precise operational semantics, strict choreographies not only identify the ideal flow through the B2Bi interaction, but also all paths for handling technical and business errors. Such a complete specification of control flow facilitates *agreement* and *system appropriateness* in several ways. Resolving ambiguity means enabling the automatic derivation of implementation artifacts. Although model-driven generation of BSIs may be too ambitious in real-world settings, such generated artifacts still may serve for testing the actual implementation or as reference implementation. In addition, an unambiguous semantics enables formal analysis which may be useful for statically checking the soundness of choreography models. Control flow defects like deadlocks or livelocks may be prevented and unimplementable models may be excluded [177,186]. Moreover, formal techniques may be leveraged for checking conformance of implementations to choreography models [46].

The downside of strict choreographies is that more complicated rules must be respected during choreography modeling which may be accessible to software engineers only. Moreover, compliance to stringent modeling rules may hinder creativity and discussion and thus even be counterproductive to communication between business experts. In addition, rules for ensuring implementability and soundness may impose limits on control flow expressiveness so that some valid types of models may unnecessarily be excluded.

Both types of choreographies need additional technical detail such as communication endpoint configuration to be filled in for implementation. This is the nature of gradually refining models and turning these into implementations. However, strict choreographies are designed such that adding detail does not necessarily change the semantics of the choreography. In so far, note that the choice between cartography and strict choreographies is a choice of modeling approach and not a choice between textual or visual models. Visual models may be defined such that their semantics is unambiguous, too (cf. chapter 6).

Both types of choreographies, cartography and strict, have valid use cases depending on the implementation scenario, development process and employees' skills. It may even be sensible to create both types of choreographies subsequently where cartography choreographies are specified first to build a common basis between business experts and strict choreographies are derived afterwards as implementation contract. Note, however, that this work is not about the optimal development process model but about the technical use of choreographies and orchestrations.

Therefore, this work focuses on the definition of strict B2Bi choreographies as motivated in the last chapter. Note that the ebBP specification itself postulates the need for strict choreographies by pointing out that *"[..] the specification of choreography definition and the Business Transaction protocol defines unambiguously which business message (DocumentEnvelope or Business Signal) is expected by any of the parties."* [134, lines 1966-1968]. This raises the question why the ebBP specification is not just used as is. Section 4.1 discusses in how far ebBP falls short in providing precise semantics for B2Bi choreographies. Section 4.2 subsequently introduces the integration architecture as representation of the execution environment. Then, section 4.3 focuses on the specification of BTs as atomic building block of B2Bi choreographies whereas sections 4.4 and 4.5 introduce and define in detail ebBP-ST and ebBP-Reg. Finally, section 4.6 introduces SeqMP as multi-party B2Bi choreography model that allows for the analysis of synchronization dependencies.

4.1. ebBP Deficiencies

The goal of ebBP is the definition of a B2Bi choreography format that is centered around the concept of BTs and BCs as defined in section 2.3. However, *"it [ebBP] is not intended to incorporate a methodology, and does not directly prescribe the use of a methodology"* [134, lines 380-381]. For the specification of strict choreographies, some methodology covering execution environment details must at least implicitly be defined. Hence, it is a natural thing that clarifications on ebBP are needed for being amenable to the purpose of this thesis. This section specifies which aspects of the ebBP specification need clarification for using it as strict choreography definition format.

At the level of BTs, six concrete BT patterns are defined that vary in their BT configurations (cf. section 2.3 for the concept of BT configuration). These patterns originate from the UMM specification where they are used for identifying different interaction scenarios between business partners. While this may have some effect on the partner internal processing of BTs, the influence on choreography and orchestration of interactions is captured in the BT configurations. The semantics of BT configurations is defined informally in [134, section 3.4.9.1] and some considerations on implementation are made. However, the description is not precise enough to uniquely determine message flow at runtime. In particular, *"how [.. QoS] parameters translate to implementation decisions is unspecified"* [134, lines 1605-1606]. Yet, the implementation of QoS parameters such as reliability or security may heavily influence message flow. Just assume that reliability cannot be assumed as a quality of the transport channel which would require the implementation of complex reliable messaging protocols at the application layer and hence significantly influence message flow.

In order to precisely specify the execution semantics of BTs, the implementation of QoS has to be taken into consideration and an execution model has to be defined on top of those considerations (cf. section 4.3).

It is vital to note that the Web Services based execution model of section 4.3 is not covered by the ebBP *Operation Mapping* for WSDL [134, section 3.4.9.8]. This mapping is based on so-called *interface operations* that are added to ebBP documents. Interface operations can be interpreted as abstract operation names that have to be mapped to actual WSDL operations later on: *"An ebBP definition does not itself contain a reference to a WSDL file, but rather references to abstract operation names, which can be de-referenced with specific WSDL files, specified at the Collaboration Protocol Profile"* [134, lines 1620-1622]. However, the definition of such syntactic relationships alone does not substitute a full-fledged execution model because behavioral aspects beyond simple request-reply interactions are left out. Furthermore, a complete Web Services based execution model can be specified without ebBP operation mappings based solely on BT configurations so that this part of the specification is not further considered.

The lack of precise assumptions about the execution environment also becomes evident in figure 13 of the ebBP specification that details the *"Computation of the Status of a Business Transaction Activity"* [134]. This specification is supposed to define the message exchange sequences for arbitrary BT configurations by defining the message receipt and send obligations of the BT requester and responder (cf. 2.3). However, the ebBP BTA status computation definition allows for at least four scenarios in which the requester and responder end up in different end states:

1. In a request-only scenario without RA and AA, the sender determines *'Success'* immediately after having sent the business document. However, the responder may end up in end state *'Failure'* if the document is detected to be invalid.
2. In a request-only scenario with a RA, assume the business document and the RA are exchanged successfully and hence the requester determines *'Success'*. Then, if the backend system does not accept the business document, the responder is supposed to end up in state *'Failure'* as given by the protocol specification.
3. In a request-response scenario with RAs, if the responder determines a timeout in state *'Wait for Receipt from Requestor'* and the message exchange is asynchronous and the sender still sends the requested RA afterwards then there is a deviating end state.
4. In a request-response scenario with RAs and AAs, if the requester determines a timeout in state *'Receive Business Signal or Business Document'* and the message exchange is asynchronous and the sender still sends the AA then there is a deviating end state.

Moreover, it is striking that there are 4 different types of results defined for the requester (*'Success'*, *'Failure'*, *'BusinessSuccess'* and *'BusinessFailure'*) whereas there are only two for the responder (*'Success'* and *'Failure'*).

At the level of BCs, the ebBP standard defines rules for composing sequences of BTs and other BCs. However, no formal execution semantics for such BCs is defined nor grammar rules that ensure soundness of resulting choreographies. Again, this is a natural thing as the precondition for defining execution semantics and soundness is the definition of an execution model so that implementability can be taken into consideration.

It is noteworthy, though, that ebBP *does* define *some* constraints on how to compose models. However, these are not always consistently defined throughout the standard. For example, in [134, line 2990] the constraint is defined that *"an XOR Fork MUST be followed with a Join where waitForAll = false"*. However, [134, lines 2012-2014] defines that *"The semantics of Fork and Join are such that for instance a Fork MAY be defined without a corresponding Join. In this case, the TimeToPerform element MUST NOT be used"*. Similarly, [134, lines 1996-1997] says that *"an XOR Fork may be designed to operate like a Decision"* where there is no constraint that a decision node must be followed by a join node. The first statement and the latter two statements cannot coexist without further *ambiguous* interpretation.

Moreover, [134, lines 2895-2906] defines constraints on the use of the *fromBusinessStateRef* and *toBusinessStateRef* attributes of *FromLink* and *ToLink* elements saying that these may not point to control flow nodes or end states. However, these constraints are broken in several examples by pointing to *Success* and *Failure* states using *toBusinessStateRef* attributes (cf. [134, page 62 and 66]). Similar syntactic constraints that complicate the representation of shared states are discussed in section 4.4.

Remember that ebBP explicitly postulates the need for atomic BT execution and for unambiguous choreography definition. How the execution environment for such choreographies and the choreographies themselves may look like is demonstrated in the remainder of this chapter.

4.2. Integration Architecture

The integration architecture presented in this section reflects the underlying approach of this work which is about implementing B2Bi choreographies by means of orchestrations. As a shared central hosting unit among B2Bi partners is frequently not available or not wanted, each integration partner is assumed to implement an orchestration process on its own that then implements the choreography in cooperation with its peers. To implement this, an underlying publication of this thesis [188] proposes a corresponding distributed integration architecture for performing ebBP choreographies. Its core characteristics are modularization and separation of control flow logic from business logic. For each ebBP BT/BC a separate set of so-called control processes (BSI implementations) handle control flow and deal with distributed computing issues. In order to handle legacy systems, business logic is assumed to be encapsulated by backend systems. The backend systems signal the need for new BT/BC executions to the control processes which in turn call back the backends' business document creation and validation facilities. The interaction between control processes and backends of an integration partner is assumed to be safe in the sense that messages do not get lost due to unreliable media or system crashes.

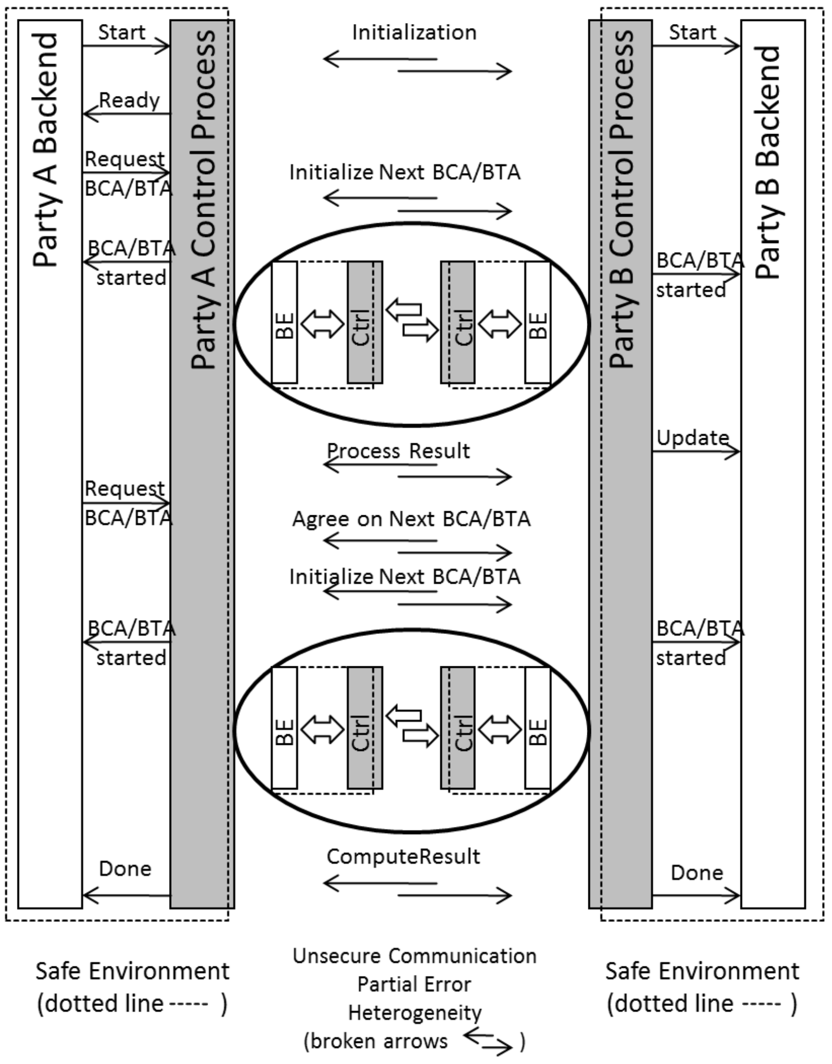


Figure 4.1.: Modularized B2Bi Scenario

Figure 4.1 shows a modularized integration scenario of two integration partners A and B. At the BC level, there is a backend component as well as a control process for each integration partner (long vertical boxes). Partner A starts out with detecting the need for performing the agreed-upon BC. A's backend correspondingly sends a *Start* message to A's control process which, in turn, initiates the BC together with B's control process. B's control process notifies B's backend that a new BC instance has been started. Subsequently, A's backend gets notified that the collaboration initialization has been finished and then requests the execution of a lower level BTA. The control processes then negotiate BTA parameters like an instance identifier or a time limit and pass on control to the lower level BTA control processes (presented

within the oval forms) which eventually produce a result value. This result value is then used for routing the control flow. In this fashion, BTAs defined in the ebBP choreography will be performed until an end state has been reached.
While this scenario leaves out several details about BC level control process interaction, it provides the principle setting for understanding the execution models of BTs as well as ebBP-ST, ebBP-Reg and SeqMP. The software artifacts for implementing the architecture as well as guidelines for deriving these are given in chapter 5.

4.3. ebBP BusinessTransaction Representation

The ebBP specification postulates the need for executing BTs as atomic units of work:

"The Business Transaction is an atomic unit of work. All of the interactions in a Business Transaction MUST succeed or each party MUST revert their state to the state prior to the start of the BTA." [134, lines 2232-2234]

An execution model for BTs must respect several challenging circumstances when implementing transactional semantics. Firstly, the incorporation of the business signals RA and AA may require tasks that cannot immediately be performed. For example, the necessary validation activities for sending an AA may require loading the exchanged business document into a business application which is not directly accessible from the BSI. This is even more true if response business documents have to be exchanged which frequently requires the involvement of human decisions. Hence, locking data items similarly to the implementation of database transactions is not acceptable. Furthermore, the distributed nature of a BT implementation has to be catered for. Integration partners are likely to operate heterogeneous integration systems so that interoperability problems have to be solved. Moreover, an adequate failure model that includes lost or duplicated messages and partial failures has to be applied. Finally, the execution of BTs in the context of superordinate BCs as described in the last section has to be respected as well as the integration setting that not only comprises the two interacting BSIs but also the backend systems.

The execution model presented here has also been presented in [192] and is defined in terms of communicating state-machines. The model is flexible, composable and QoS-aware as it allows for different BT configurations (cf. section 2.3.1), enables composition and reflects the implications of Web Services based QoS realization. Although implementability and validity have been checked for Web Services and BPEL by means of a prototype (see section 5.2), the model is abstract in allowing for different communication technologies. The basic paradigm of the model is to consider the business documents of a BT as exchanged and valid if and only if all defined business documents and business signals of the BT have been exchanged, the communicating partners have achieved agreement about that and all configured QoS parameters have been applied.

The next two sections first discuss the requirements for defining an execution model in more detail and then present the actual model.

4.3.1. Requirements Analysis

The core requirement for a BT execution model is to be compliant with the ebBP specification of a BT as presented in section 2.3.1 which includes the configuration of business documents, business signals and QoS. Additional requirements concerning result computation and realization of QoS, configuration options of the ebBP model as well as the surrounding BC's control flow are described in sections 4.3.1.1, 4.3.1.2 and 4.3.1.3.

4.3.1.1. Mutual Agreement and QoS

Technically speaking, the very purpose of executing BTs is aligning state between integration partners. If a BTA is performed within a collaboration then the result of the BTA must be mutually agreed upon once the BTA execution has terminated. Otherwise, the state of integration partners may diverge. Clearly, the business implications of a BT execution depend on the business logic applied to the exchanged documents. As ebBP only defines technical aspects of BTs, the model used in this work defines the result of a BT execution as the *exchange state of business documents and business signals as well as fulfillment of ebBP QoS requirements*. Fulfillment of ebBP QoS requirements is necessary because these may affect the exchange state of a business document. For example, if authentication is required then a business document must not be considered to have been successfully exchanged in case authentication information is missing. Furthermore, QoS attributes have to be implemented in a mutual way, i.e., both requester and responder must be authenticated, sure about integrity et cetera. Finally, QoS has to be equally applied to business documents and business signals because business signals influence the result of a BT execution. For example, using an unauthenticated RA to acknowledge legibility of an authenticated business document is not sensible.

4.3.1.2. Configurability

Different integration scenarios have different requirements with respect to state alignment and QoS. Transmitting RA and AA signals as well as defining strict security requirements may be required for the exchange of a purchase order while a RA may be sufficient for the exchange of a catalog. The different ebBP business transaction patterns [134, section 3.4.9.1] like *Notification*, *QueryResponse* or *CommercialExchange* having different QoS needs give testament to this fact. The patterns originate from the UMM standard and reflect different types of integration scenarios, but for the purpose of this thesis these are simply distinguished by means of their different BT configurations (cf. section 4.1). The rationale of this concept is that the BT configurations are the result of the analysis of the integration scenarios in UMM. Thereby, some of the configuration parameters are optional. For example, the *Notification* pattern requires using a requesting BA as well as a RA signal while the use of an AA as well as some QoS attributes are optional.

The ebBP *DataExchange* pattern is conceived by ebBP as an extensibility pattern and hence is the most flexible pattern in allowing either one or two BAs and defining no constraints upon the use of RAs, AAs or QoS parameters. In so far, the remaining ebBP business transaction patterns can be interpreted as instantiations of the *DataExchange* parameterization options. Therefore, supporting the *DataExchange* pattern is selected as a requirement for the ebBP BT execution model.

4.3.1.3. Composition Context

The invention of the *isConcurrent* parameter for ebBP BTs shows that its context, i.e., the way a BT is used within BCs, influences the BTs' implementation. Considering the integration setting described in section 4.2, the following requirements can be derived:

Concurrent execution At one point in time, more than one instance of a BT could be active.

Multiple execution Concurrent execution is one form of executing a BT several times. Moreover, a BT may be repeatedly executed in a loop, in sequence or in different control flow branches. This implies that routing of the collaboration needs more information than simply protocol *success* or *failure*. In order to enable flexible routing, a BT implementation should not only propagate a generic protocol outcome to the superordinate collaboration but also hand over the business documents exchanged.

Reconfiguration Multiple execution of BTs within a BC (using BTAs) implies that execution parameters may vary as well. The ebBP model provides *isConcurrent*, *hasLegalIntent* and *timeToPerform* as configuration options at the BTA level.

4.3.2. Execution Model

This section presents the execution model for ebBP BTs by defining communicating state machines for BT control processes. The model is composable, flexible, QoS-aware and abstract.

Composability is facilitated in two ways. Firstly, a consistent BT execution result with respect to the state of business document/signal exchanges and the fulfillment of QoS is ensured. Secondly, requirements emerging from the composition context as described in section 4.3.1.3 are respected. Furthermore, the execution model is *flexible* by allowing for almost arbitrary instance values of the *DataExchange* parameters. Moreover, the model is *QoS-aware* as the influence of QoS realization (sec. 4.3.2.1) on control flow (sec. 4.3.2.2) explicitly has been respected, most notably whether or not mutual agreement upon message delivery can be expected to be available at the messaging level. In so far, section 4.3.2.1 only analyzes QoS realization up to the point that is decisive for building the control flow model. Finally, the execution model is *abstract* in not prescribing a concrete implementation of control processes. This is due to the fact that the implementation of control processes highly depends on the IT landscape of integration partners. Although a Web Services and BPEL-based prototype has been implemented, partners may not want to implement control processes on top of BPEL or not even use Web Services. Other communication technologies that meet the assumptions about QoS realization may be chosen as well.

4.3.2.1. Realization of Quality-of-Service

Realization of ebBP QoS requirements necessitates the combination of distributed algorithms targeting security or reliability goals. Simply implementing a security layer on top of a reliability layer is not possible for two reasons:
1) Mutual realization of security-related QoS properties may create a new reliability problem. In practice, security properties like authentication or integrity are frequently implemented by means of attaching asymmetric digital signatures to the message payload. The receiver then can verify the authenticity of the sender, but the sender cannot be sure about authenticity of the receiver. Sending a signed acknowledgment with a hash value of the original message does not help because then the sender of the acknowledgment cannot be sure who has received the acknowledgment.
2) A malicious attacker must be assumed. A malicious attacker basically may try to manipulate any message exchanged between integration partners. This not only holds true for the message payloads but for lower-level transport messages as well. This means that an attacker may try to manipulate communication by means of tampering with unsecured reliability messages. While an attacker may not be able to break arbitrary security goals like that, the reliability property is endangered.

Put short, mutual realization of security and reliability properties calls for algorithms that implement a secured reliable messaging layer.
The complexity of such algorithms raises the question which implementation components should perform QoS algorithms. In general, these algorithms may be implemented at the protocol level, i.e., by the control processes, or at the messaging level, i.e., by the messaging technology used for communication. Note that control processes are application level protocols. In practice, the complexity of distributed algorithms barely is acceptable at the application level.
Fortunately, realization of QoS at the messaging level is available for Web Services, especially the combination of reliable messaging and security. [4] and [19] both report the successful verification of the so-called "Secure WS-ReliableMessaging Scenario". In particular, [4] report successful verification of *"mutual authentication between client and service on all security-relevant messages."* If Web Services are used, reliability (ebBP attribute *isGuaranteedDeliveryRequired*), confidentiality (*isConfidential*), integrity (*isTamperDetectable*) and authentication (*isAuthenticated*) therefore can be assumed to be implementable at the messaging level. The same holds true for authorization (*isAuthorizationRequired*) that can easily be implemented once that authentication is available.
Concurrency (*isConcurrent*) cannot be implemented at the messaging level as it concerns the separation of BT execution instances as well as synchronization of access

to backend systems. If BPEL is used for control process implementation then BPEL correlations [137, section 9] can be used for enabling concurrent control process instances. If not, WS-Addressing [224] could be used. Finally, if neither BPEL correlations nor WS-Addressing are appropriate, a BPEL correlations like mechanism can be implemented at the application level. As regards the synchronization of access to backends the assumption is made that functionality that is exposed to integration processes must be able to deal with concurrency. If necessary, the backends can be informed about the *isConcurrent* value by means of a flag.
hasLegalIntent cannot directly be implemented because its semantics are not clearly defined (cf. section 2.3.1). As far as BT control processes are concerned, integration partners may choose to map the *hasLegalIntent* value to different instantiations of other ebBP QoS attributes. Moreover, the *hasLegalIntent* value should be passed on to the backend applications.
isNonRepudiationRequired and *isNonRepudiationReceiptRequired* are special in implying a very hard error model. Non-repudiation usually is defined as the property that the sender of a particular message cannot deny having sent the message. The attempt to deny having sent/received a message implies that an integration partner cannot be assumed to behave as defined in a protocol specification. If so, the possibility of implementing two-way non-repudiation is questionable, i.e., the sender cannot deny having sent a message while the receiver cannot deny having received it. Consistently, no WS-* standard could be found that implements non-repudiation in a mutual way. Therefore, non-repudiation is proposed to be implemented in an asymmetric way by simply attaching a signature to business documents and business signals and archiving these messages upon arrival. Once the non-repudiated message has been archived the receiver can claim to have successfully received the message. If a BT execution succeeds (which should be the standard case) then having implemented non-repudiation in an asymmetric way does not do any harm. If it fails, the receiver of the non-repudiated message may assert a claim based on the message. The remaining QoS features listed in table 2.1 can easily be implemented at the control process level and are discussed in the next section.

4.3.2.2. Control Flow

This section presents the control flow of control processes using a state machine based model. A core design decision is the choice of communication style between automata, i.e., synchronous or asynchronous. [24] find that assuming synchronous communication allows for easier automata design than asynchronous communication. Consistently, synchronous communication has frequently been assumed for model design and analysis (for example [15, 107, 235, 241]). The last section spelled out that the realization of QoS properties also is possible for this communication style. Therefore, the work at hand also adopts a synchronous communication model. Note that synchrony as used here only concerns one single message exchange, i.e., the sender of a message only blocks until the message is delivered. Processing of the

message is then performed asynchronously. Technically speaking, this corresponds to messaging with a buffer length of 0.
From a software engineering point of view, assuming synchronous communication between control processes imposes more strict requirements in terms of availability and throughput than asynchronous communication. This limitation can be justified because the control processes' only task is controlling the interaction between integration partners. At the same time, control processes can be used to implement decoupling between the integration partners' backend systems. For example, incoming business documents could be stored in a message queue by the control process and then be picked up by the backend when appropriate.

State Machine Model This paragraph describes the control processes based on the running example of listing 2.4 on page 54. The paragraph *Configurability* on page 98 illustrates how to derive control processes for different instances of an ebBP BT configuration. For formalization, the notion of a *control process state machine* (CPS) is defined as follows:

Definition 4.3.1 (Control Process State Machine)
A control process state machine *(CPS) is a 7-tuple (S,s_0,F,E,I,G,δ) consisting of the following elements:*

- *S a finite set of states and $s_0 \in S$ the initial state.*
- *$F \subseteq S$ a non-empty set of final states.*
- *$E = M \cup L$ a set of events where M is a set of message exchange events and L is a set of local events. Every $m \in M$ consists of a communication partner p, a direction $d \in \{!, ?\}$ and the actual message type t. '$p!t$' denotes an outgoing message t to partner p and $p?t$ denotes an incoming message t from partner p.*
- *I a set of counters with domain $\mathbb{N}_0$.*
- *G a set of guards with $G \subseteq \mathbb{N}_0 \times \{<, \leq, =, >, \geq\} \times I$*
- *δ is a partial transition function $\delta : S \times E \times 2^G \rightharpoonup S \times 2^I$*

□

The following communication partners are used for the definition of CPSs that can be combined pairwise, i.e., the communication between two CPSs is not visible to a third CPS. REQ and RES denote the requestor and responder control process implementing an ebBP BT. BE_1 and MA_1 denote the backend and the master process of the requestor party where BE_1 implements business logic like validation of business documents while MA_1 implements the superordinate ebBP BC. Correspondingly, there are BE_2 and MA_2 roles for the responder party. In addition, RAC denotes a component implementing business document legibility validation (e.g., schema validation) which is needed by the responder control process. The state diagrams depicted in figures 4.2 and 4.3 denote the transition relation of the REQ and RES CPSs.

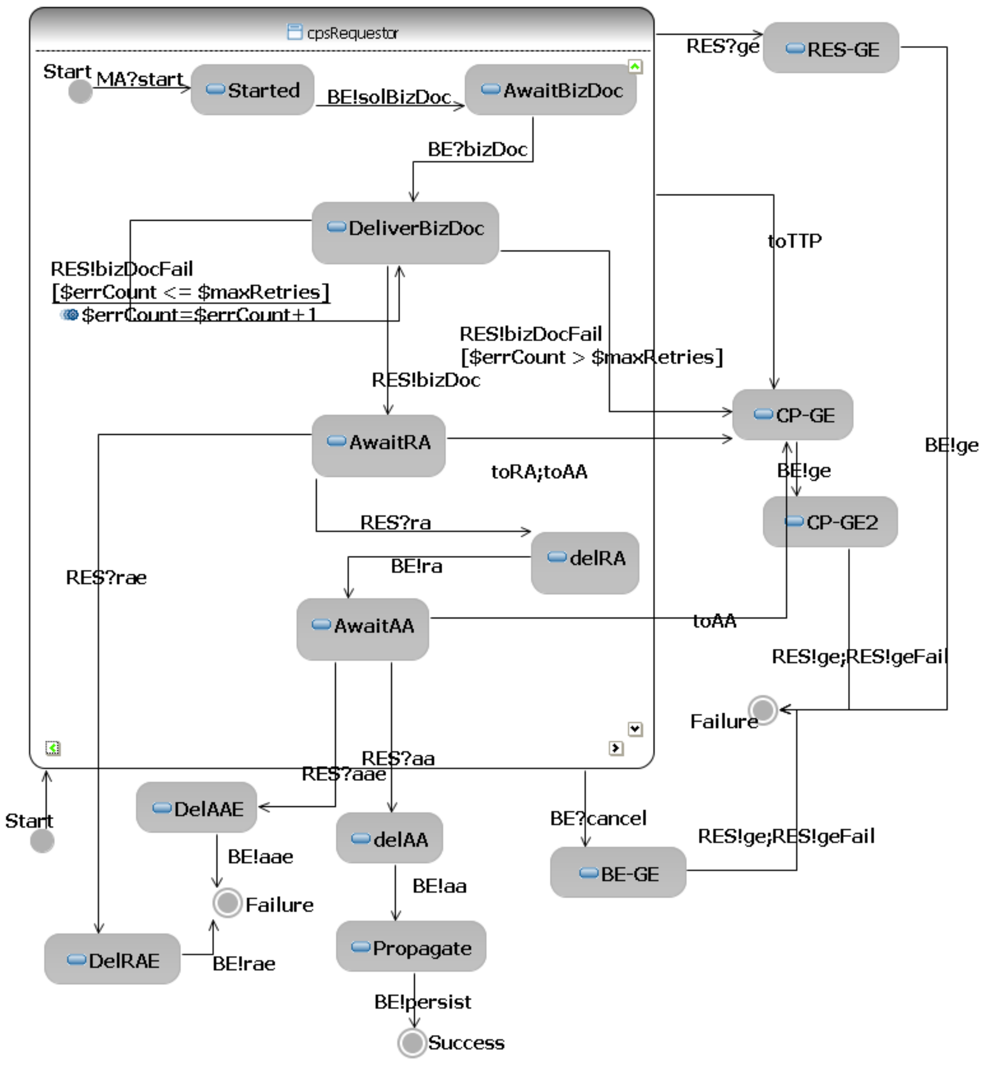


Figure 4.2.: Requester Control Process Machine

Note that the integration of control processes with backend systems is private logic of the integration partners and therefore is not a formal element of the ebBP BT execution model. Nonetheless, the BE_k, MA_k and RAC parties are included for showing one valid way of integration. Clearly, any implementation that accepts the same set of communication sequences between control processes is acceptable.
The message types (set M of a CPS) selected for interaction between control processes are aligned with the ebBP specification:

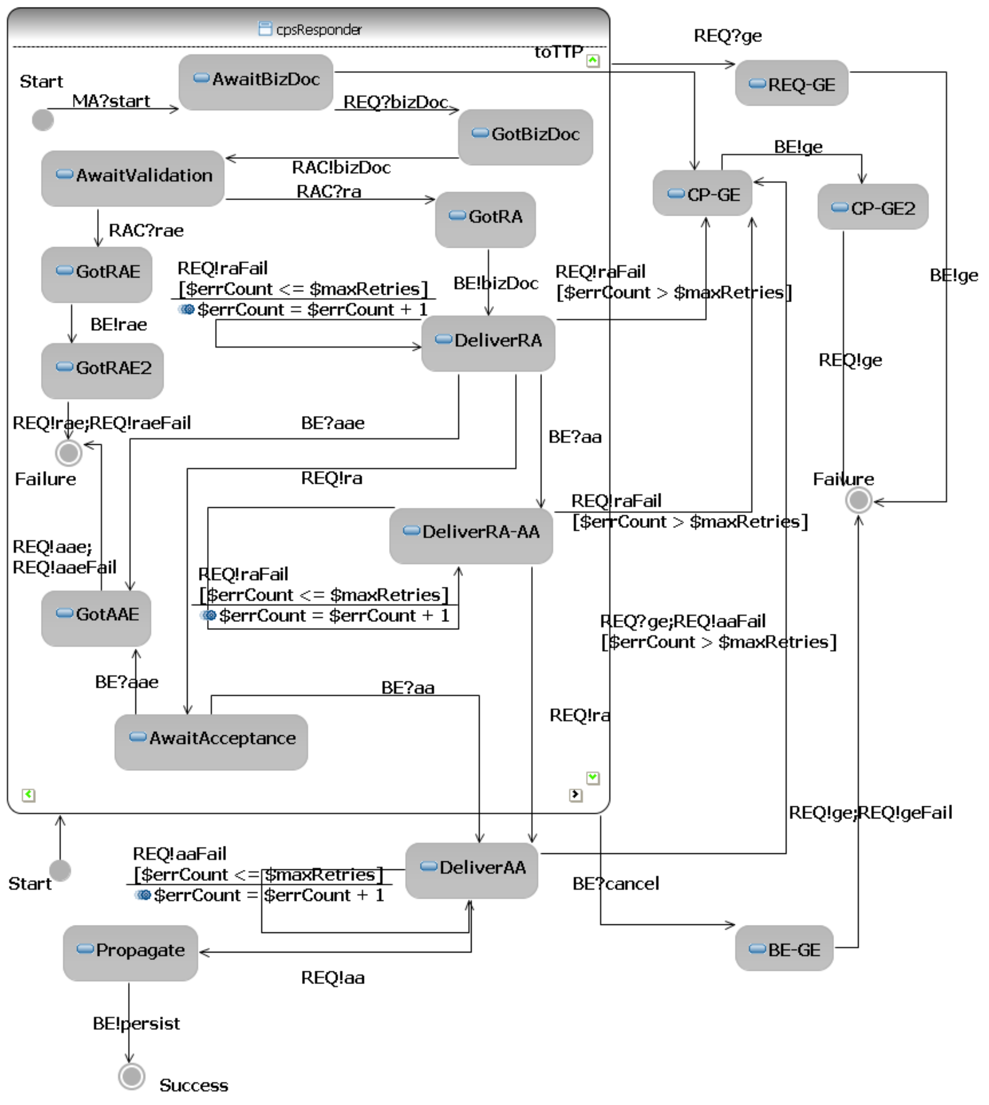


Figure 4.3.: Responder Control Process Machine

- *bizDoc* is an abstract type denoting the business document to be exchanged.
- *ra* and *aa* denote ebBP's ReceiptAcknowledgement and AcceptanceAcknowledgement, respectively, while *rae* and *aae* denote the corresponding exceptions.
- *ge* denotes GeneralException that is used by ebBP for conveying exception information that is not covered by *rae* and *aae*.
- *start* (containing initialization information), *solBizDoc* (for soliciting the business document to be exchanged), *cancel* (for giving the backend the opportunity to cancel the BT run) and *persist* (for specifying that the message exchange succeeded) are only needed for modeling interaction with the remaining parties.

The following local events (set L of a CPS) have been defined. For any message type m $\in$ M, there is a <*m*>*Fail* event that models the case that the sending CPS could not deliver the message of type *m* to the receiving CPS. The event is modeled as local because the receiving CPS might not even realize that the message exchange failed. There are no <m>Fail events defined for the interaction between control processes and BE_k, MA_k, RAC processes because this interaction is assumed to be safe (cf. section 4.2). *toTTP*, *toRA*, *toAA* model timeout events for the *timeToAcknowledgeReceipt*, *timeToAcknowledgeAcceptance* and *timeToPerform* parameters of an ebBP BT definition. Basically, these timeout events are controlled by the REQ CPS in order to avoid concurrent timeouts. Only in the *AwaitBizDoc* state of the RES CPS, a *toTTP* event is defined in order to avoid that the RES CPS waits forever. *I* is defined as {errCount, maxRetries} for implementing ebBP's *retryCount* parameter. *isIntelligibleCheckRequired* is not explicitly reflected in the control flow. ebBP provides Receipt/AcceptanceAcknowledgements and the corresponding exception types for conveying legibility information. The details of the validation checks required are to be defined by the integration partners. The *Success* and *Failure* states in figures 4.2 and 4.3 represent the state machines' final states. These states represent purely technical results, i.e., whether all necessary messages successfully have been exchanged or not. As a business transaction implementation may be reused in several contexts (cf. section 4.3.1.3), the computation of a business level result, e.g., whether or not a document exchange represents the obligation to pay an invoice, is to be done at the level of the calling collaboration implementation. This implies that the protocol result together with the exchanged business document are propagated to the calling implementation upon reaching a final state (not explicitly modeled in the state machines).
For presentation purposes, two visual simplifications have been introduced in figures 4.2 and 4.3 that are not reflected in the CPS definition. Composite states have been introduced for limiting the number of transitions. The actual automaton can be derived by adding a copy of each transition emerging from a composite state to every (non-pseudo-) state contained in the composite state. Moreover, when a transition carries two events separated by a semicolon, then there actually are two transitions defined (and guards and increments only apply to the last event). Finally, note that the names of states do not contribute to the semantics of the state machines and therefore are not explicitly explained.

Semantics Common state machine semantics do not assume a synchronous communication model (like in [38]) or do not consider local events (like in [15]). Therefore, the semantics of a CPS is defined operationally as follows (remember that the model definition is non-hierarchical):
Consider a pair of CPS machines (CPS_1,CPS_2). Let M_p be the set of all message types that can be exchanged between CPS_1 and CPS_2. Furthermore, let a configuration c be defined as $c \in \mathcal{C} = \mathrm{S}_1 \times \mathrm{E}_1^* \times \mathrm{V}_1 \times \mathrm{S}_2 \times \mathrm{E}_2^* \times \mathrm{V}_2$ where V is the set of all possible values of the counters of a CPS: V$= 2^{I \to \mathbb{N}_0}$. Finally, let $\psi : \mathrm{G} \times \mathrm{V} \to \{tt, ff\}$ be

the boolean function that evaluates guards under given counter values to true (tt) or false (ff) and ϕ the function that extracts the communication partner from a messaging event. The semantics of a pair of CPS machines CPS_1 and CPS_2 is then defined by the relation $\vdash$ on $\mathcal{C} \times \mathcal{C}$ that captures communication between automata (*comm*) or local behavior of one automaton (*local*). Rules 1 and 2 describe a message exchange from CPS_1 to CPS_2 and a local event of CPS_1, respectively. For brevity, symmetric rules for CPS_2 are left out:

$$
\begin{array}{ll}
1: & (s_1, CPS_2!m_1\upsilon_1, \mathrm{v}_1, s_2, CPS_1?m_2\upsilon_2, \mathrm{v}_2) \vdash^{comm} \\
 & (s_1', \upsilon_1, \mathrm{v}_1', s_2', \upsilon_2, \mathrm{v}_2') \;\; iff \\
 & m_1 = m_2 \in \mathrm{M}_p \wedge \\
 & \delta(s_1, CPS_2!m_1, \alpha_1) = (s_1', \beta_1) \wedge \\
 & \alpha_1 = \{\} \wedge \forall i \in \beta_1 : \mathrm{v}_1'(i) = \mathrm{v}_1(i) + 1 \wedge \\
 & \delta(s_2, CPS_1?m_2, \alpha_2) = (s_2', \beta_2) \wedge \\
 & \alpha_2 = \{\} \wedge \forall i \in \beta_2 : \mathrm{v}_2'(i) = \mathrm{v}_2(i) + 1
\end{array}
$$

$$
\begin{array}{ll}
2: & (s_1, e_1\upsilon_1, \mathrm{v}_1, s_2, e_2\upsilon_2, \mathrm{v}_2) \vdash^{local} \\
 & (s_1', \upsilon_1, \mathrm{v}_1', s_2, e_2\upsilon_2, \mathrm{v}_2) \;\; iff \\
 & (e_1 \in L_1 \vee \phi(e_1) \neq CPS_2) \wedge \\
 & \delta(s_1, e_1, \alpha_1) = (s_1', \beta_1) \wedge \\
 & \forall g \in \alpha_1 : \psi(g, \mathrm{v}_1) = tt \wedge \\
 & \forall i \in \beta_1 : \mathrm{v}_1'(i) = \mathrm{v}_1(i) + 1
\end{array}
$$

Configurability Section 4.3.1 mandates that the ebBP BT execution model should be freely configurable. The implementation of the execution parameters as listed in table 2.1 has been discussed above. Yet, there could be different selections of control messages, i.e., whether Receipt/AcceptanceAcknowledgements are to be used or if a second business document is to be exchanged. The correct CPSs when leaving out a Receipt/AcceptanceAcknowledgement can be derived by first removing all transitions triggered by an *ra/rae/aa/aae* and then removing all states that do not have outgoing transitions triggered by message exchanges anymore and then adjusting dangling transitions correspondingly. For example, if RA and AA would have to be removed from the REQ CPS, the *RES!bizDoc* transition would have to be connected to the *Propagate* state. The correct REQ CPS for an ebBP two-action BT can be derived by appending the RES state machine to the REQ state machine. When appending, the *Start* state of the RES CPS replaces the *Propagate* state of the REQ state machine. The two-action RES state machine then can be derived correspondingly. Note that the responsibility for monitoring the overall *timeToPerform* then must be switched from the REQ CPS to the RES CPS after finishing the first *BusinessAction*.
Configurability also raises the question of dependencies between configuration parameters. For example, defining a *timeToAcknowledgeReceipt* requires that a Re-

ceiptAcknowledgement is exchanged. A discussion of such dependencies is provided in [172] that has been contributed to during the dissertation project of this thesis.

4.4. ebBP-ST Choreographies

ebBP Shared State (ebBP-ST) choreographies have been proposed in chapter 3 as choreography class that particularly targets *comprehensibility* and *state-based modeling* by explicitly modeling so-called shared states. The motivation for explicitly representing state is the insight that the very purpose of B2Bi interactions is the alignment of the partner systems' states. The basic concept for using state in choreographies has been proposed in [186] which is a conceptual predecessor of this thesis leveraging UML activity diagrams as choreography representation. There, the concept of so-called *shared states* concertedly reached/left by integration participants is used for explicitly modeling the state changes realized by performing BTs. However, ebBP does not naturally support the explicit representation of state. This section shows how state can explicitly be represented in ebBP[1].

The presentation begins with clarifying why the explicit modeling of shared states is beneficial:

- **Shared states explicitly capture the effect of performing BusinessTra nsactions.**
  This is helpful for communication and agreement among personnel of different enterprises. For example, the sequences of BusinessTransaction executions that lead to a valid contract document can be deduced from analyzing the execution paths that lead to an explicitly modeled shared state *Contract*.

- **Shared states enable intelligible communication of progress.**
  Monitoring of B2Bi processes requires intelligible communication of progress. An executive overview of progress can be achieved more easily by signaling the current shared state, e.g., whether a *Quote* or *Contract* state has been reached, than by signaling the sequence of BusinessTransaction executions together with the business documents exchanged and the business document evaluation rules for assessing the implications of a business document's content.

- **Shared states allow for the specification of distributed timeouts.**
  B2Bi is the connection of business processes that run on scarce resources. Consistently, resource reservations frequently have a limited time horizon. Release of resources without requiring successful execution of BusinessTransactions can be specified by attaching timeouts to shared states. For example, a shared state *Quote* could be specified to be left after 3 days if no BusinessTransactions lead to a follow-on state.

- **Shared states provide natural synchronization points for specifying control flow.**
  The selection of BusinessTransactions that are admissible at a particular point in time typically depend on the integration partners' system states, e.g., whether a valid contract document is available or not. It is easier to control the execution

[1]published as [160, 182]

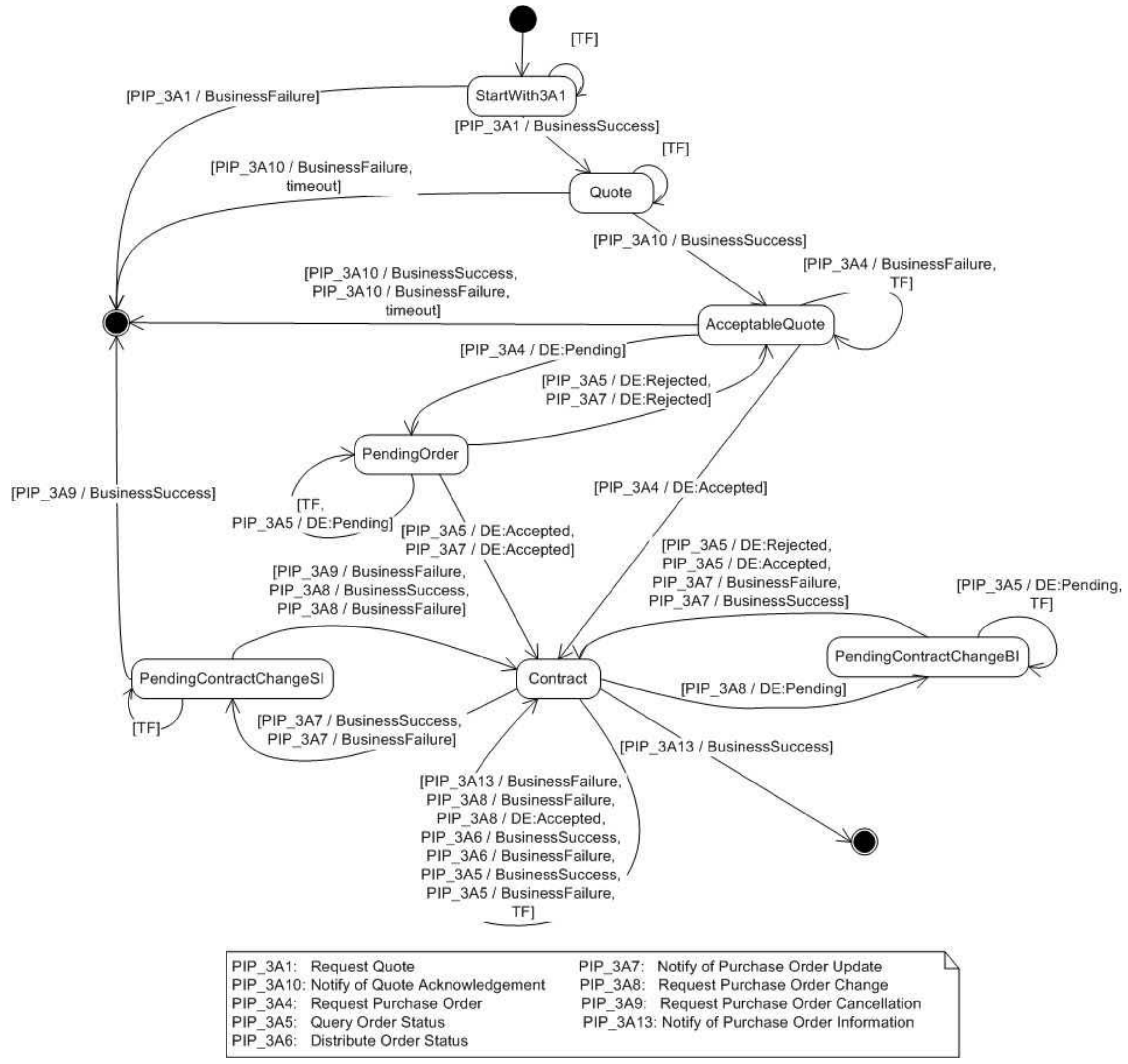
paths that emerge from an explicitly modeled shared state than controlling all execution paths that continue all possible execution paths that lead to a particular state.

As shared states are not naturally supported by ebBP, a workaround for representing shared states in an ebBP compliant way is presented first. In order to allow for more straightforward and intuitive modeling, an ebBP schema extension for modeling shared states is described as well. Based on the concept of shared states, ebBP-ST is introduced as modeling style for a restricted set of ebBP collaborations that enable real-world size B2Bi processes. The class of collaborations that comply with this style is concisely characterized by a formalization of shared state based ebBP models. This formalization also lays the foundation for describing the conversion of shared states modeled by means of an ebBP schema extension into ebBP compliant models, for precisely capturing the meaning of shared state based collaborations by means of an operational semantics, and as foundation for specifying a distributed BPEL-based implementation of ebBP models. A prototypic translation engine that generates BPEL implementations from ebBP choreographies has been developed by Christoph Pflügler as part of his diploma thesis which has been supervised in the context of this work's dissertation project. This prototype proves the realizability of the operational semantics defined and a real-world sized use case is applied for evaluating practical relevance. The interested reader is referred to [160, 182] for details on the generation of BPEL processes from ebBP-ST. In addition, the translation algorithm is given in appendix B.

The rest of this chapter is organized as follows. Section 4.4.1 presents the use case for evaluating ebBP-ST and exemplifies the benefits of shared state based modeling. An ebBP schema compliant XML model of shared states as well as the corresponding schema extension based model is presented in section 4.4.2. Moreover, an intuitive description of valid shared state-based collaborations is given that is formalized in section 4.4.3. The formalization is accompanied by an algorithm for converting shared state-based collaborations modeled by means of the ebBP extension into ebBP compliant models as well as the operational semantics of shared state-based collaborations. The evaluation of ebBP-ST is presented in section 4.4.4 by discussing the results of implementing the proposed integration architecture and an ebBP2BPEL translator, by analyzing the complexity of the algorithms presented, and by investigating the complexity reduction achieved by using the ebBP schema extension for shared states instead of the ebBP compliant workaround.

4.4.1. Use Case

The use case for evaluating ebBP-ST is based on RosettaNet PIPs. PIPs, classified in clusters like cluster 3 Order Management and segments like segment 3A Quote and Order Entry, describe the application context, the content and the parameters for the electronic exchange of one or two business documents. A use case consisting


[TF]
StartWith3A1
[PIP_3A1 / BusinessFailure]
[PIP_3A1 / BusinessSuccess]
[TF]
Quote
[PIP_3A10 / BusinessFailure, timeout]
[PIP_3A10 / BusinessSuccess]
[PIP_3A4 / BusinessFailure, TF]
[PIP_3A10 / BusinessSuccess, PIP_3A10 / BusinessFailure, timeout]
AcceptableQuote
[PIP_3A4 / DE:Pending]
[PIP_3A5 / DE:Rejected, PIP_3A7 / DE:Rejected]
PendingOrder
[PIP_3A9 / BusinessSuccess]
[TF, PIP_3A5 / DE:Pending]
[PIP_3A5 / DE:Accepted, PIP_3A7 / DE:Accepted]
[PIP_3A4 / DE:Accepted]
[PIP_3A9 / BusinessFailure, PIP_3A8 / BusinessSuccess, PIP_3A8 / BusinessFailure]
[PIP_3A5 / DE:Rejected, PIP_3A5 / DE:Accepted, PIP_3A7 / BusinessFailure, PIP_3A7 / BusinessSuccess]
[PIP_3A5 / DE:Pending, TF]
PendingContractChangeBI
PendingContractChangeSI
Contract
[PIP_3A8 / DE:Pending]
[TF]
[PIP_3A7 / BusinessSuccess, PIP_3A7 / BusinessFailure]
[PIP_3A13 / BusinessSuccess]
[PIP_3A13 / BusinessFailure, PIP_3A8 / BusinessFailure, PIP_3A8 / DE:Accepted, PIP_3A6 / BusinessSuccess, PIP_3A6 / BusinessFailure, PIP_3A5 / BusinessSuccess, PIP_3A5 / BusinessFailure, TF]

PIP_3A1:	Request Quote	PIP_3A7:	Notify of Purchase Order Update
PIP_3A10:	Notify of Quote Acknowledgement	PIP_3A8:	Request Purchase Order Change
PIP_3A4:	Request Purchase Order	PIP_3A9:	Request Purchase Order Cancellation
PIP_3A5:	Query Order Status	PIP_3A13:	Notify of Purchase Order Information
PIP_3A6:	Distribute Order Status		

Figure 4.4.: Use Case for Evaluating the ebBP-ST

of nine shared states and nine PIPs has been created exemplifying shared state-based modeling. The RosettaNet document type definitions have been imported by means of ebBP *BusinessDocuments* and their flow has been remodeled using ebBP BT. How a PIP can be represented as an ebBP BT has been contributed to and described in [172]. The use case is taken from RosettaNet PIP segment 3A (*Quote and Order Entry*) and models a process for negotiating a contract. The size of standard processes as defined by the Northern European Subset (cf. `http://www.nesubl.eu/`, last access: 12/20/2011) (NES) is comparable to the use case of this section, so the use case's size can be considered to be practically relevant.

The use case represents a binary collaboration. The two business partners take the roles of *buyer* and *seller* throughout the whole collaboration. The overall goal of the composition is the negotiation of a contract and of contract changes. The collaboration terminates as soon as the buyer has received the goods and services he requested. The description of the use case starts with explaining the usage of the selected PIPs.

PIP 3A1: Request Quote
This PIP is used to start the collaboration. The buyer requests a quote for some particular goods or services. The seller answers with a response document either representing a *BusinessSuccess* or a *BusinessFailure*. (ebBP allows for associating the *BusinessSuccess* or *BusinessFailure* result values with types of response documents). In the former case the seller may reserve resources for the buyer, but the buyer is not obliged to accept the quote. In the latter case the collaboration is terminated immediately.

PIP 3A10: Notify of Quote Acknowledgement
If the buyer has received a valid quote she is obliged to use this PIP to inform the seller whether the quote is generally acceptable or not. If not, the collaboration is terminated immediately. Otherwise the seller extends the reservation of resources and waits for an order. The buyer is still not obliged to accept the quote.

PIP 3A4: Request Purchase Order
This PIP can either be used to start the collaboration or to sign a contract after having confirmed a quote to be acceptable with the help of PIP 3A10. The buyer sends a quote to the seller that can be answered with either an *Accepted*, a *Rejected* or a *Pending* message in corresponding ebBP `DocumentEnvelopes` (ebBP allows for capturing the result of a BTA by referring to the DocumentEnvelope types of the exchanged messages). If the answer is *Accepted* the parties have signed a legally binding contract. If the answer is *Rejected* the collaboration terminates immediately. If the answer is *Pending* the buyer waits until the seller notifies her about the decision using PIP 3A7 or the buyer queries the decision with PIP 3A5.

PIP 3A5: Query Order Status
The buyer can use this PIP if there is a valid contract, if the decision of the seller about a quote (PIP 3A4) is still pending, or if the decision of the seller about a contract change request (PIP 3A8) is still pending. The answer of the seller has to be evaluated depending on which of these situations applies.
In the first case, the seller can either provide new information about order progress or just tell that no progress has been achieved. In case of the other two situations, the seller can either send an *Accepted*, a *Rejected* or a *Pending* message. If an order has not yet been decided upon, a new contract is signed (*Accepted*), the collaboration is terminated immediately (*Rejected*) or the decision is further postponed (*Pending*). If a contract change request has not yet been decided upon, either the current contract is replaced by a new one (*Accepted*), the current contract remains valid (*Rejected*) or the decision is further postponed (*Pending*).

PIP 3A6: Distribute Order Status If there is a valid contract, the seller can use this PIP to communicate information about order progress to the buyer.

PIP 3A7: Notify of Purchase Order Update The seller must trigger this PIP if she has sent a *Pending* message in PIP 3A4 or 3A8 before. The seller may reply using an *Accepted* or a *Rejected* message. By analogy with PIP 3A5, a new contract

is then signed (*Accepted*) or the collaboration is terminated/the current contract remains valid (*Rejected*).
Furthermore, the seller can use PIP 3A7 to request contract changes. As PIP 3A7 is a Single-Action Activity, i.e., only one business message can be exchanged, the buyer cannot directly answer such a request. To answer a contract change request, the buyer must either use PIP 3A8 or PIP 3A9.
PIP 3A8: Request Purchase Order Change. This PIP is usually used by the buyer to request a contract change. The seller can then either send *Accepted* to confirm the change, *Rejected* to keep the current contract or *Pending* to postpone the decision.
Furthermore, this PIP is used to answer a contract change request initiated by the seller with PIP 3A7. If the buyer wants to reject the change request, she sends a *Purchase Order Change Request* message that exactly contains the data of the current contract. The current contract then remains valid no matter what the seller answers. To be concise, the seller should send a *Rejected* message. If the buyer is about to accept the change request of the seller (with modifications), she sends a *Purchase Order Change Request* message (with modifications). The seller may then only respond with an *Accepted* message to sign a new contract or with a *Rejected* message to keep the current contract.
PIP 3A9: Request Purchase Order Cancellation
This PIP is usually used by the buyer to cancel a current contract. Moreover, the buyer can offer the cancellation of a contract instead of a contract change requested by the seller (PIP 3A7).
In both cases the seller can only send an *Accepted* message to rescind the contract or a *Rejected* message to keep the current contract.
PIP 3A13: Notify of Purchase Order Information
The buyer uses this PIP to notify the seller about processing updates or the fulfillment of the contract.

The use case is visualized in a state machine-like manner in figure 4.4. The start of the collaboration is represented by the unique start element. Each shared state is represented as a state and the executions of PIPs as BTAs are represented as transitions. Note that, for presentation purposes, this representation deviates from the actual semantics defined below where a BTA is a special type of control flow state that consumes time. The event part of a transition in figure 4.4 is used to name the BTA (PIP) to be executed and the guard part of a transition is used to capture the outcome of BTAs. As decisions are not explicitly visualized, there may be multiple transitions for the same shared state that are triggered by the same event. The condition guards of the particular transitions, however, are mutually exclusive. The permissible ebBP guard values for the use case are *AnyProtocolFailure* (denoted `TF`), *BusinessFailure* or *BusinessSuccess*. *AnyProtocolFailure* captures arbitrary technical problems during performing BTAs. If no such problems occur, *BusinessSuccess* indicates that integration partners did achieve their goals from a business point of view whereas *BusinessFailure* indicates they did not. Finally, guard values based on

DocumentEnvelopes (denoted with a leading `DE:`) that relate to the content of the latest business document exchanged using suitable XPath expressions are allowed as well. Two final states are used to represent an ebBP Failure state (on the left-hand side) and an ebBP Success state (on the right-hand side). Although the execution of `PIP_3A9` in state `PendingContractChangeSI(SellerInitiated)` may terminate with a *BusinessSuccess* guard value, it still represents a failure from the overall collaboration perspective.

Using this use case the benefits of shared state based modeling can easily be demonstrated:

Shared states explicitly capture the effect of performing BusinessTransactions.

Obviously, the business contents exchanged in BTAs govern the state alignment actions to be performed in participating integration systems. Shared states can be used to represent the result achieved by having exchanged corresponding content. For example, PIP 3A4's request document may be answered by different DocumentEnvelopes inidicating *Pending*, *Accepted* or *Rejected*. Using shared states, collaboration partners can express that a *Pending* message results in a *PendingOrder* state and not in state *Contract*.

Shared states allow for intelligible communication of progress.

Process visibility and analysis are fostered by communication of progress information about active business processes. As the effect of BTAs may depend on the actions taken previously, communicating the type and content of a BTA is not necessarily sufficient for uniquely determining progress. In the use case scenario, having performed PIP 3A5 with result *DE:Pending* may lead to shared states *PendingOrder* or *PendingContractChangeBI*, depending on the previous state.

Shared states allow for the specification of distributed timeouts.

B2Bi collaborations may necessitate the reservation of resources, for example, in shared states *Quote* and *AcceptableQuote* of the use case. By attaching timeout values to these shared states, a time limit for resource release may be defined in case collaboration partners do not trigger BTAs in a timely manner.

Shared states provide natural synchronization points for specifying control flow.

As BTAs depend on and modify system state, shared states are a natural way for specifying control flow, i.e., the sequences of business document exchanges that lead to the same state and the business document exchanges that require the same state as precondition. For example, assume that the collaboration depicted in figure 4.4 has progressed to state *Contract*. Then, the sequence of PIP 3A8 and PIP 3A5 (change initiated by the buyer role) may lead to a new contract as well as the sequence of PIP 3A7 and 3A8 (change initiated by the seller role). Conversely, PIP 3A5 is only admissible in state *PendingContractChangeBI* but not in *PendingContractChangeSI*. In case such commonalities need to be expressed, alternative modeling approaches without constructs for joining/splitting control flow may lead to far more complex models. Figure 4.5 depicts an alternative model for the states *StartWith3A1*, *Quote*,

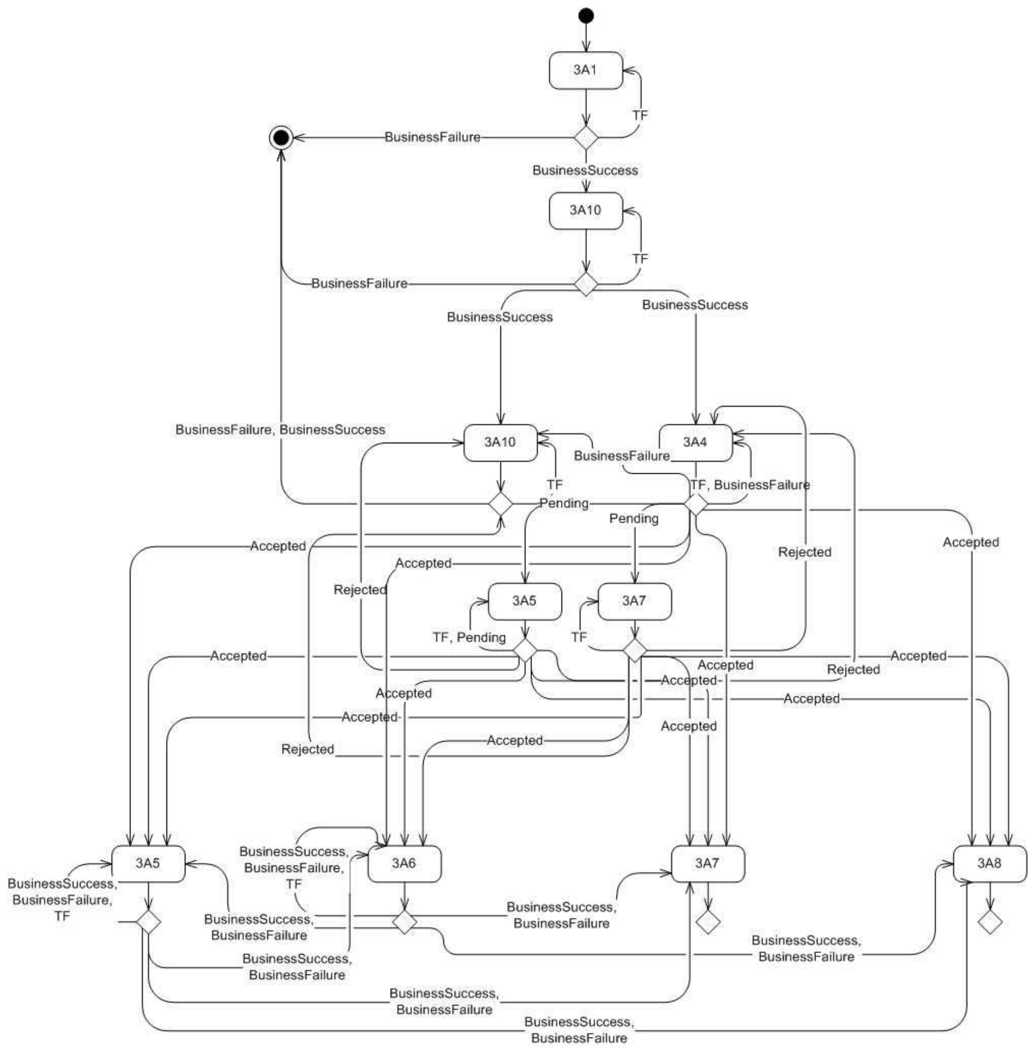


Figure 4.5.: Alternative Modeling of an Excerpt of the Use Case Demonstrating Control Flow Explosion

AcceptableQuote and *PendingOrder* of figure 4.4 only. In figure 4.5, control flow is specified only by connecting BTAs (rounded boxes) and decision nodes (diamonds) and attaching guards to the transitions. The complexity explosion due to removing shared states can be explained by the fact that an extra transition must be introduced for each BTA that is allowed for in a particular shared state. For example, BTAs 3A10 and 3A4 are admissible in *AcceptableQuote*. Therefore, after having performed 3A10 successfully (in shared state *Quote*), two paths have to be specified for allowing both 3A10 AND 3A4. Note that extra control flow links also are a result of different possible BTA results and BTAs that are applicable in different states. For example, there are two links from BTA 3A4 to a 3A7 node in figure 4.5. This is due to the fact that 3A7 may be performed in state *PendingOrder* (i.e., result *Pending* for 3A4) as well as in state *Contract* (i.e., result *Accepted* for 3A4).

An obvious way for working around this situation without shared states is using pure control flow pseudo-nodes like ebBP `Join` for merging alternative paths and ebBP `Fork` for splitting up alternative paths. Therefore, one would only have to create transitions from the ebBP `Decision` nodes to the matching `Join` nodes and then using `Fork` nodes as needed for splitting up control flow again. Unfortunately, this is not permitted by ebBP because transitions may only reference BTAs or BCAs (cf. toBusinessStateRef/fromBusinessStateRef constraints in [134, sec. 3.8.2]).

4.4.2. Informal ebBP Models

A shared state of a B2Bi collaboration as introduced in [186] is a synchronization point that represents alignment of information items among integration partners and captures the progress of a collaboration. Integration partners use BTAs to consistently align information and thus concertedly leave and reach shared states. While the preceding sections motivated the use of shared states, this section discusses the modeling of shared states using ebBP.

4.4.2.1. ebBP Compliant Shared State Model

There is no ebBP construct that directly matches the concept of a shared state so these have to be emulated. Generally speaking, a state can be modeled with an ebBP *Join* construct followed by a *Fork* construct. However, ebBP prohibits directly linking *Joins* and *Forks* as the corresponding *fromBusinessStateRef* and *toBusinessStateRef* attributes may only reference BTAs or BCAs [134, sec. 3.8.2]. To overcome this constraint in a standard compliant manner, a workaround can be used. The concept of an *EmptyBTA* based on the extensible ebBP transaction type *DataExchange* is introduced that serves as a target for linking to a shared state and for connecting the *Join* and *Fork* of a shared state.

Listing 4.1 shows the ebBP representation of the shared state *Quote* (cf. above and figure 4.4). The *EmptyBTA* before the shared state's *Join* is used as target of ebBP *Decisions* that are not allowed to directly link to *Joins* [134, sec. 3.8.2]. The shared state's *Join* links to another *EmptyBTA* that is connected to the shared state's *Fork*. This *Fork* then specifies a *ToLink* for every BTA that is permissible to be performed from this shared state. Shared state timeouts, i.e., the point in time when shared states should be left without performing a BTA, can also be specified on this *Fork*. In case such a timeout occurs, it has to be switched to the *EmptyBTA* defined in the *ToLink* that carries the corresponding *timeout ConditionExpression*.

Employing two *EmptyBTAs* allows for different semantics when linking to a shared state with respect to timeouts: In case of linking to the *EmptyBTA* before a shared state, its timeout is reset whereas linking to the *EmptyBTA* within the shared state does not have this effect. The latter case is particularly useful if protocol failures occur during performing a subsequent BTA which means that the shared state actually has not been left. Note that ebBP *Joins* and *Forks* are only used for modeling states and are not allowed elsewhere in the collaboration description.

Listing 4.1: ebBP Compliant Model of a Shared State

```

<!-- State Quote -->
<BusinessTransactionActivity businessTransactionRef="empty"
    nameID="empty_before_Quote">
 <TimeToPerform></TimeToPerform>
 <Performs currentRoleRef="Buyer" performsRoleRef="empty1"/>
 <Performs currentRoleRef="Seller" performsRoleRef="empty2"/>
</BusinessTransactionActivity>

<Join waitForAll="false" nameID="Quote">
 <FromLink fromBusinessStateRef="empty_before_Quote"/>
 <FromLink fromBusinessStateRef="empty_before_Quote"/>
 <ToLink toBusinessStateRef="
     empty_in_Quote"/>
</Join>

<BusinessTransactionActivity businessTransactionRef="empty"
    nameID="empty_in_Quote">
 <TimeToPerform></TimeToPerform>
 <Performs currentRoleRef="Buyer" performsRoleRef="empty1"/>
 <Performs currentRoleRef="Seller" performsRoleRef="empty2"/>
</BusinessTransactionActivity>

<Fork nameID="fork_Quote" type="XOR">
 <TimeToPerform duration="P3D"/>
 <FromLink fromBusinessStateRef="empty_in_Quote"/>
 <ToLink toBusinessStateRef="BTA_3A10_NotifyOfQuoteAck"/>
 <ToLink toBusinessStateRef="BTA_3A10_NotifyOfQuoteAck"/>
 <ToLink toBusinessStateRef="empty_before_FAILURE">
  <ConditionExpression
    expressionLanguage="XPath1"
   expression="timeout"/>
 </ToLink>
</Fork>
```

4.4.2.2. ebBP Extension for Shared States

The motivation for providing an ebBP extension and hence dropping compliance is complexity reduction. In the workaround presented in section 4.4.2.1, four different control flow nodes have to be used for specifying a shared state and its outgoing transitions. Using the proposed extension, the same information can be represented by only one node. The new `SharedState` construct (depicted in listing 4.2) that has been defined is similar to an ebBP `Fork` node with the *type* attribute set to *XOR*.

Listing 4.2: Extension-based Model of a Shared State

```

<!-- State Quote -->
<SharedState nameID="Quote">
 <TimeToPerform duration="P3D"/>
 <FromLink
  fromBusinessStateRef="DECISION_3A10_NotifyOfQuoteAck"
  stTimeoutReset="false">
  <ConditionExpression
   expressionLanguage="ConditionGuardValue"
   expression="AnyProtocolFailure" />
 </FromLink>
 <ToLink toBusinessStateRef="BTA_3A10_NotifyOfQuoteAck" />
 <ToLink toBusinessStateRef="Collaboration_FAILURE">
  <ConditionExpression
    expressionLanguage="XPath1"
   expression="timeout" />
 </ToLink>
</SharedState>
```

It reuses the ebBP definitions of `TimeToPerform`, `FromLink` and `ToLink`, but removes the cardinality constraints on the number of `FromLinks` and `ToLinks` completely. Thus, a logical link between any control flow node and a `SharedState` can syntactically either be specified within the `SharedState`, within the control flow node under consideration, or using a separate ebBP `Transition` element. For enabling the distinction between resetting a timer when linking to a shared state or not, the optional boolean flag `stTimeoutReset` has been added to `FromLinks` and `ToLinks`. Note that the proposed ebBP schema extension for representing share states does not render existing ebBP models obsolete. Only the ebBP constraint requiring *fromBusinessStateRef* and *toBusinessStateRef* attributes to exclusively reference BTAs or BCAs [134, sec. 3.8.2] is dropped. ebBP neither describes the rationale behind that constraint nor defines a semantics that relies on that constraint. Section 4.4.3 defines a semantics for shared state based ebBP collaborations that works without that constraint.

4.4.2.3. Shared State-based Collaborations

This section informally describes the class of shared state-based ebBP collaborations (ebBP-ST) proposed for B2Bi process specification. Basically, a shared state is entered by reaching the *EmptyBTA* before the shared state. The *Fork* of the shared state then links to all BTAs that are permissible for the respective shared state. Each of these BTAs (except *EmptyBTAs*) must be followed by an ebBP *Decision* that evaluates the outcome of the BTA. Predefined ebBP *ConditionGuardValues* and user-defined *DocumentEnvelopes* are used for determining the follow-on shared state of a *Decision*. In case an ebBP *AnyProtocolFailure* is detected, the *Decision* must link back to the *EmptyBTA* within the shared state the BTA to be evaluated was started from. Otherwise, it is linked to the *EmptyBTA* before the same or another shared state.
The restrictions chosen are aligned with two goals. Firstly, a large part of real-world processes should be representable in a straightforward manner. Secondly, distributed BPEL implementations should be automatically derivable from the specified collaborations.

In a multi-case study, [166] report the results from an investigation of 16 business processes from six Dutch organizations: *"One of the striking observations was that out of the 16 processes considered none of these processes incorporated concurrent behavior, i.e. parallel processing of single cases. Business processes turned out to be completely sequential structures. Their routing complexity was only determined by choice constructs and iterations."* This finding is further backed by the B2Bi models created for the eBIZ-TCF project (`http://www.moda-ml.net/moda-ml/repository/ebbp/v2008-1/en/`, last access: 12/20/2011) that also do not specify concurrent behavior. In so far, ebBP-ST can be assumed to cover a large part of real-world processes.

Achieving the second goal, that means, deriving BPEL-based implementations from ebBP-ST is shown in appendix B.

Informally, ebBP-ST can be characterized as a subset of the class of multi-transmission interactions as defined in [11] with the special restriction that only two collaboration partners are allowed. Before a formal model of ebBP-ST will be presented, the main characteristics are summarized:

- A choreography is modeled as a single ebBP *BusinessCollaboration*. Hierarchical compositions are not supported.
- Only binary collaborations are supported, i.e., the number of integration partners within the collaboration is limited to two.
- A collaboration starts with an ebBP *Start* that immediately links to the initial shared state of the collaboration.
- ebBP *Decisions* are only allowed directly after BTAs.
- Alternative paths are realized by ebBP *Decisions* and by ebBP *Forks* used for representing shared states.
- Looping is realized by *Decisions* that link back to shared states that have been visited before.
- The only case in which a *Decision* branch does not link to a shared state is when process termination is detected. In this special case a *Decision* links to an EmptyBTA before an ebBP *Success* or *Failure* state.
- A choreography ends when a final state, i.e., an ebBP *Success* or *Failure* state is reached. Multiple *Success* and *Failure* states are allowed per collaboration. As multiple instances of BTAs are not allowed for and as state is synchronized after each BTA (there are only two participants), a choreography immediately ends when a final state is reached.
- At any point in time, there is at most one active BTA (no multiple instances). In order to ensure this, ebBP *Forks* have to set the *type* attribute to *XOR* and ebBP *Joins* must have the *waitForAll* attribute set to *false*.
- An EmptyBTA may only link to one single ebBP *Join* or *Fork* element.

The formalization of this informal description is given next.

4.4.3. Formal ebBP models

The formalization of the informally defined class of shared state-based ebBP collaborations serves several purposes. Above all, it is used for precisely defining the set of valid models that is accepted for translation into distributed BPEL implementations. Furthermore, it is used as concise basis for describing the translation of shared states modeled using the ebBP-ST schema extension into ebBP compliant shared states.

Therefore, the formalization closely reflects the main different ebBP element types for specifying `BusinessCollaborations`. In the following, 'STBC' will be used to abbreviate a *shared state based ebBP BusinessCollaboration*. STBCs that contain shared states modeled by means of the schema extension will be denoted ESTBC (Extension-type STBC) whereas STBCs that employ ebBP complaint shared states will be denoted WSTBC (Workaround-type STBC).
Finally, as ebBP does not provide a formal semantics, another major motivation for providing a formalization is an unambiguous and comprehensible description of STBC semantics. Section 4.4.3.1 introduces the formalization of WSTBCs and section 4.4.3.2 presents the semantics. Section 4.4.3.3 then discusses the formalization of ESTBCs and an algorithm for translating ESTBCs to WSTBCs.

4.4.3.1. WSTBC

Definition 4.4.1 (WSTBC)
A workaround-type shared state-based ebBP BsinessCollaboration *(WSTBC) is an extension of a directed graph defined as a 5-tuple* (R,N,G,ϕ,θ) *consisting of the following elements:*

- $R = \{r_1,r_2\}$ *is the set of collaboration participant roles where* r_1 *is statically declared to be the* leader *of the collaboration.*
- $N = \{s_0\} \cup FORK \cup JOIN \cup DEC \cup SBTA \cup SEBTA \cup T$ *is a set of nodes where the components of the union are pairwise disjoint.*
  - s_0 *is the initial node.*
  - T *being the non-empty set of terminal nodes.*
  - *FORK is a set of ebBP* ***`Fork`*** *elements with the* type *attribute set to 'XOR'.*
  - *JOIN is a set of ebBP* ***`Join`*** *elements with the* waitForAll *attribute set to 'false'.*
  - *DEC is a set of ebBP* ***`Dec`*** *elements.*
  - *SBTA is a set of ebBP* ***`BusinessTransactionAcitivities`****.*
  - *SEBTA is a set of* EmptyBTAs *as described in section 4.4.2.1.*
- $G = (L \times E_L) \cup \{(XPath1, \text{'timeout'})\} \cup \{tt\}$ *is a set of guards defined as either a pair of a language* $l \in L$ *and expression* $exp \in E_l$ *defined in l, the special purpose pair (XPath1, 'timeout') for specifying a shared state's timeout, or the boolean constant true (tt).*
  Out of the admissible ebBP expression languages, ***`ConditionGuardValue`*** *(CGV) and* ***`DocumentEnvelope`*** *(DE) are supported where* E_{CGV} *is an enumeration of generic ebBP protocol outcomes and* E_{DE} *is the set of ebBP* ***`DocumentEnvelopes`*** *defined for the directly preceding BusinessTransaction.*

- *The function $\phi : FORK \rightarrow \mathbb{N}_0 \cup \{-1\}$ that assigns a timeout value to every* `Fork` *node. '-1' is used for denoting an undefined timeout value.*
- *θ is a transition relation $\theta : N \times 2^G \times N$ with the constraint that θ is the union of the following components:*
  - $\rightarrow^{Start} \subseteq \{s_0\} \times \{tt\} \times SEBTA$ *and* $\left|\rightarrow^{Start}\right| = 1$.
  - $\rightarrow^{ST1} \subseteq SEBTA \times \{tt\} \times JOIN$
  - $\rightarrow^{ST2} \subseteq JOIN \times \{tt\} \times SEBTA$
  - $\rightarrow^{ST3} \subseteq SEBTA \times \{tt\} \times FORK$
  - $\rightarrow^{Terminal} \subseteq SEBTA \times \{tt\} \times T$
  - $\rightarrow^{Trigger} \subseteq FORK \times \{tt\} \times SBTA$
  - $\rightarrow^{Eval} \subseteq SBTA \times \{tt\} \times DEC$
  - $\rightarrow^{Update} \subseteq SBTA \times \{tt\} \times SEBTA$
  - $\rightarrow^{Route} \subseteq DEC \times 2^G \setminus \{(XPath1, timeout)\} \times SEBTA$
  - $\rightarrow^{Timeout} \subseteq FORK \times \{(XPath1, timeout)\} \times SEBTA$

  Note that all components of θ except for $\rightarrow^{Trigger}$ are partial functions $N \times 2^G \rightharpoonup N$. Furthermore, a workaround-based shared state (cf. section 4.4.2.1) may result in two identical elements $t_1 = t_2 = (n_1, \{tt\}, n_2)$ that would have to be added to either $\rightarrow^{ST1}$ or $\rightarrow^{Trigger}$. In that case, t_2 is discarded. □

Two nodes n_k, n_l are *directly connected* in a WSTBC if there is a triple (n_k,g,n_l) or (n_l,g,n_k) $\in \theta$. For (n_k,g,n_l), n_k *directly precedes* n_l and n_l *directly follows* n_k. Furthermore, two nodes n_k, n_l are *connected* in a WSTBC if there is a triple (n_k,g,n_l) or (n_l,g,n_k) $\in \theta^*$ where θ^* is the reflexive-transitive closure of θ. A sequence of nodes $[n_1, .., n_x]$ is defined such that for any i, $1 \leq i < x$: $\exists(n_i, g, n_{i+1}) \in \theta$.

Moreover, several functions defined on the components of a WSTBC are needed (for later use in the translation algorithms):

- $in : N \rightarrow 2^N$ computes the set of input nodes of a particular node n_i in WSTBC such that $in(n_i) = \{x | \exists(x, g, n_i) \in \theta\}$.
- $out : N \rightarrow 2^N$ computes the set of output nodes of a particular node n_i in WSTBC such that $out(n_i) = \{x | \exists(n_i, g, x) \in \theta\}$.
- $requestor : SBTA \rightarrow R$ determines which of the collaboration participant roles takes the ebBP `RequestingRole` of a particular BTA.
- $responder : SBTA \rightarrow R$ determines which of the collaboration participant roles takes the ebBP `RespondingRole` of a particular BTA.

In the definitions so far there is no shared state construct. That is due to the fact that the ebBP elements that make up a shared state are explicitly emulated. This is necessary for being able to provide a concise conversion algorithm from ESTBCs to WSTBCs. The following definition characterizes shared states within WSTBCs.

Definition 4.4.2 (Shared State)
A shared state *(ST) is defined for a WSTBC as 5-tuple $ST_{WSTBC}(e_b,j,e_i,f,\theta_{ST})$ such that*

- $e_b \in SEBTA$
- $j \in JOIN$
- $e_i \in SEBTA$
- $f \in FORK$
- $\theta_{ST} = \{(e_b, \{tt\}, j), (j, \{tt\}, e_i), (e_i, \{tt\}, f)\} \subset \theta$
- $\nexists (n_k, g, n_l) \in \theta \setminus \theta_{ST} : n_k = j \vee n_l = j \vee n_l = f$

A WSTBC is said to contain a ST$_{WSTBC}$ if it conforms to the above definition. For dealing with STs, some additional functions are needed. Let SH$_{WSTBC}$ be the set of all ST$_{WSTBC}$(e$_b$,j,e$_i$,f,θ_{ST}) contained in a WSTBC. Then, the following functions are defined:

- *nodeb, nodej, nodei, nodef, trans, nodeset* are functions on SH$_{WSTBC}\times$ N that compute the first, second, third, fourth, fifth or union of the first four components of a given ST$_{WSTBC}$.
- *parentST* : $N \to SH_{WSTBC} \cup \{\perp\}$ computes the ST$_{WSTBC}$ for a given node (the existence of such a function follows from fact 4.4.2).
- *ctrlFlow* computes the *control flow* relation ϑ of a WSTBC such that $\vartheta = \theta \setminus \bigcup_{st \in SH_{WSTBC}} trans(st)$.
- (n_k, g, ST) denotes that there is a $(n_k, g, n_l) \in \vartheta \wedge n_l \in nodeset$(ST) and
- (ST, g, n_l) denotes that there is a $(n_k, g, n_l) \in \vartheta \wedge n_k \in nodeset$(ST).

Now, the ebBP language restrictions are presented that reflect the rationale of ST based modeling. The first restriction says that *EmptyBTAs* that link to a `Fork` or `Join` always are part of a shared state ST.

Restriction 4.4.1 (ST linking)
From definition 4.4.2 it is already clear that, for a particular $st \in SH_{WSTBC}$, there are no elements in ϑ that link to $nodej(st)$ or $nodef(st)$ or from $nodej(st)$. Moreover:

- *Iff for any $ebta \in SEBTA$,*
  $\exists (ebta, tt, j) \in \theta : j \in JOIN \Rightarrow \exists st \in SH_{WSTBC} : ebta = nodeb(st)$.
- *Iff for any $ebta \in SEBTA$,*
  $\exists (ebta, tt, f) \in \theta : f \in FORK \Rightarrow \exists st \in SH_{WSTBC} : ebta = nodei(st)$.

The following fact clarifies that `Forks` and `Joins` exclusively are used for modeling STs.

Fact 4.4.1 (No Joins/Forks outside STs)
$\forall n \in JOIN \cup FORK : n \in \bigcup_{st \in SH_{WSTBC}} nodeset(st)$

Proof 4.4.1
From definition 4.4.1, and in particular the definition of θ it is clear that for all nodes e that directly precede *a node $n \in JOIN \cup FORK$, holds: $e \in SEBTA$. From restriction 4.4.1 follows that for all $e \in SEBTA$ that* directly precede *a node $n \in JOIN \cup FORK$:*
$e \in \bigcup_{st \in SH_{WSTBC}} nodeset(st)$. The fact then follows from the definition of θ and the definition of STs (def. 4.4.2). $\square$

The next fact points out that STs in a WSTBC do not overlap, i.e., a structure as depicted in figure 4.6 is forbidden.

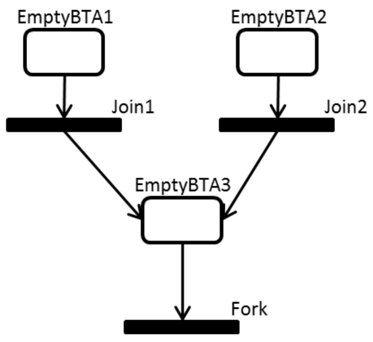


Figure 4.6.: Invalid: Shared State Overlap

Fact 4.4.2 (Disjoint STs)
For any st_1, $st_2 \in SH_{WSTBC}$ such that $st_1 \neq st_2$ holds:
$nodeset(st_1) \cap nodeset(st_2) = \{\}$

Proof 4.4.2
Assume the opposite for st_1, $st_2 \in SH_{WSTBC}$. If $nodeb(st_1) \neq nodeb(st_2)$ then $nodeset(st_1) \cap nodeset(st_2) = \{\}$ because of the definition of ST (def. 4.4.2) and $\rightarrow^{ST1}, \rightarrow^{ST2}$ and $\rightarrow^{ST3}$ being partial functions.
Similarly, if $nodeb(st_1) = nodeb(st_2)$ then $nodeset(st_1) = nodeset(st_2)$. $\square$

EmptyBTAs are used for solving ebBP schema constraints only. This implies that *EmptyBTAs* shall not contain any business logic and therefore shall always link to exactly one successor node.

Fact 4.4.3 (No Logic in EmptyBTAs)
$\forall e \in \mathit{SEBTA} : \forall (e, \{tt\}, n_k), (e, tt, n_l) \in \theta : n_k = n_l$

Proof 4.4.3
Directly follows from the definition of $\rightarrow^{ST1}, \rightarrow^{ST3}$ *and* $\rightarrow^{Terminal}$ $\square$

The following language restriction highlights that in case a ST can be left by a timeout then the following *EmptyBTA* shall precede a terminal node or be part of a different ST.

Restriction 4.4.2 (Leaving ST by timeout)
$\forall (f, g, e) \in \theta, f = nodef(st_x), st_x \in SH_{WSTBC}, f \in \mathit{FORK},$
$e \in \mathit{SEBTA}$ *holds:*
$g = (\mathit{XPath1}, \mathit{timeout}) \wedge (\exists (e, tt, t) \in \rightarrow^{Terminal} \vee$
$(e \in nodeset(st_y), st_y \in SH_{WSTBC} \wedge st_y \neq st_x))$

The next language restriction makes clear that a particular BTA may not be triggered from different STs. If the same ebBP BusinessTransaction were to be admissible in different STs then multiple BTAs of that type would have to be specified. A model that contains a structure like that depicted in figure 4.7 is invalid.

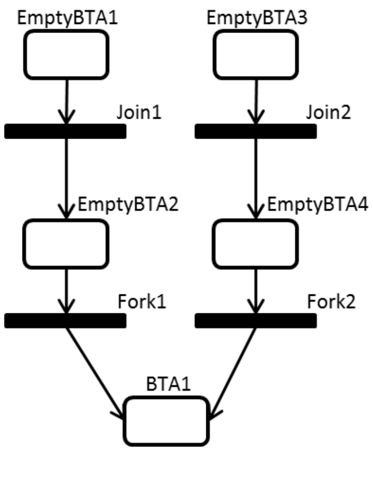


Figure 4.7.: Invalid: BTA triggered from different STs

Restriction 4.4.3 (Unique source ST of a BTA)
$\forall (f_k, tt, b), (f_l, tt, b) \in \rightarrow^{Trigger} : \; f_k = f_l$

The function $btaSrc : \mathrm{BTA} \rightarrow SH_{WSTBC}$ is used to compute the shared state st with $(nodef(st), tt, b) \in \rightarrow^{Trigger}$.

The result of a BTA determines the next ST of a collaboration. Consistently, the control flow routing decision for determining the next ST shall either be explicitly represented in the ebBP definition or the ST shall not be left. This is ensured by facts 4.4.4 and 4.4.5 as well as restrictions 4.4.4 and 4.4.5 that are presented next.

Fact 4.4.4 (Unique BTA result processing)
$\forall b \in BTA : \forall (b, tt, n_k), (b, tt, n_l) \in \theta : n_k = n_l$

Proof 4.4.4
Directly follows from the definition of $\rightarrow^{Eval}$ *and* $\rightarrow^{Update}$. □

Restriction 4.4.4 (Unprocessed BTA result)
$\forall (b, tt, e) \in \rightarrow^{Update}: e \in nodeset(btaSrc(b))$

In order to be clear which BTA's result a Decision node processes, the following restriction is defined.

Restriction 4.4.5 (Unique BTA reference)
$\forall (b_k, tt, d), (b_l, tt, d) \in \rightarrow^{Eval}: b_k = b_l$

Fact 4.4.5 (Unique source ST of a Dec)
$\forall (f_k, tt, b_m), (f_l, tt, b_n) \in \rightarrow^{Trigger}$ *and*
$(b_m, tt, d), (b_n, tt, d) \in \rightarrow^{Eval}$ *holds:* $f_k = f_l$

Proof 4.4.5
Directly follows from restrictions 4.4.3, 4.4.5 and fact 4.4.4. □

The function $decSrc : \mathrm{DEC} \rightarrow SH_{WSTBC}$ is used to compute the shared state st with $(nodef(st), tt, b) \in \rightarrow^{Trigger}$ and $(b, tt, d) \in \rightarrow^{Eval}$.

Apart from explicit representation of routing rules in collaborations, the requirement of state alignment for reaching new STs is important. The next restriction says that in case an error is detected during performing a BTA, a new state may not be reached.

Restriction 4.4.6 (No ST exit without alignment)
$\forall (d, g, e) \in \rightarrow^{Route}$ *holds:*
$g = (CGV, exp) \wedge exp$ *'indicates a protocol failure'* $\Rightarrow$
$e \in nodeset(decSrc(d))$

Finally, if the result of a BTA has been agreed upon then there is no room for non-determinism to decide about the next ST to reach which is highlighted by fact 4.4.6.

Fact 4.4.6 (Unique BTA result)
For any result of a given BTA b *with*$(b, tt, d), d \in DEC$*, holds:*
$\forall (d, g_k, e_k), (d, g_l, e_l) \in \rightarrow^{Route}: g_k = g_l \Rightarrow e_k = e_l$

Proof 4.4.6
Directly follows from the definition of $\rightarrow^{Route}$ □

Figure 4.8 visualizes the main control flow options for STBCs when respecting all restrictions. Now that the main syntactical constraints of WSTBCs have been defined, the WSTBC execution semantics can be made precise.

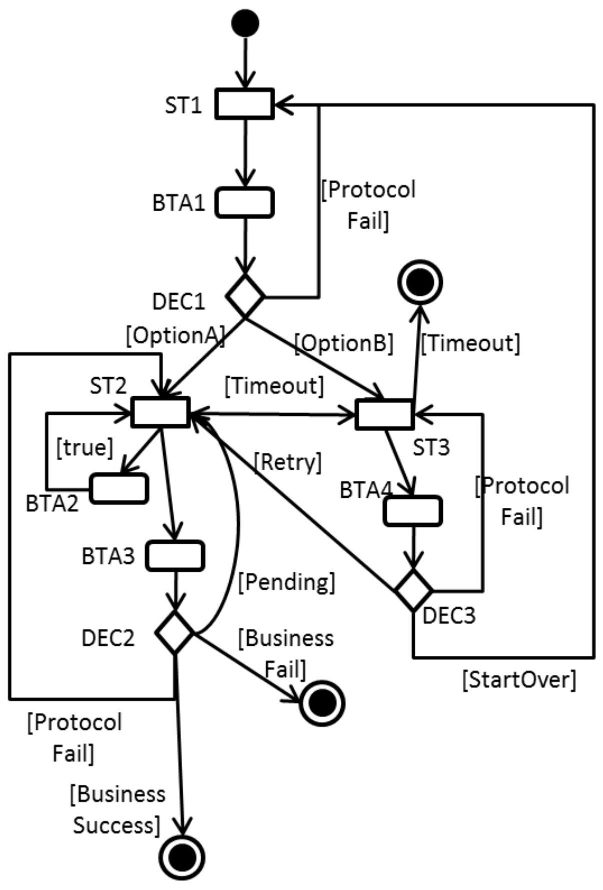


Figure 4.8.: Valid Example WSTBC

4.4.3.2. WSTBC Execution Semantics

ebBP does not define a formal semantics on its own. So, the execution of BTAs and BCAs has to be defined. The details of BTA execution are formalized in section 4.3 and therefore are not repeated for the formalization of this section. So, the assumption is used that performing a BTA takes some time and eventually either leads to an agreed-upon result or a protocol failure. Essentially, the following semantics describes how to iteratively perform the BTA execution model of section 4.3 as defined by the links between shared states, BTAs and *Decisions*.
Let $\mathcal{C} \in N \times O \times V_t \times V_{th}$ be the configuration of a WSTBC where N is the set of nodes as in definition 4.4.1, O is the set of all possible outcomes of all BTAs of a WSTBC, V_t represents all possible timer values, and V_{th} represents all possible timer threshold values. As t is defined to be a discrete timer, let the domain of V_t and V_{th} be $\mathbb{N}_0 \cup \{-1\}$. Note that although there are multiple timeouts defined (for different STs), at one point in time, there is at most one timer active. Let χ be the function that computes the outcome of a BTA that has just finished. Furthermore, $\psi : 2^G \times N \times O \times V_t \times V_{th} \rightarrow \{true, false\}$ is the function that evaluates a given set of guards under a given configuration to one of the boolean constants true or

false. In particular, $\psi(tt, \mathcal{C}) = true$ for arbitrary $\mathcal{C}$. Any element $t = (n_k, g, n_l) \in \theta$ is said to be *enabled* for a given $\mathcal{C} = (n_c, o, v_t, v_{th})$ iff $n_c = n_k \wedge \psi(g, \mathcal{C}) = true$. It directly follows that all $t = (n_k, g, n_l) \in \rightarrow^{Start} \cup \rightarrow^{ST1} \cup \rightarrow^{ST3} \cup \rightarrow^{Terminal} \cup \rightarrow^{ST2} \cup \rightarrow^{Trigger} \cup \rightarrow^{Eval} \cup \rightarrow^{Update}$ are always enabled once a configuration $\mathcal{C}$ contains n_k as the first component.
The semantics is defined operationally by the relation $\vdash \subseteq (N \times O \times V_t \times V_{th}) \times (N \times O \times V_t \times V_{th})$. The initial configuration of a WSTBC is $\mathcal{C} = (s_0, \{\}, -1, -1)$, where -1 for the timer and timer threshold values indicates that there is neither a current timer nor a corresponding threshold. All transitions in $t = (n_k, g, n_l) \in \rightarrow^{Start} \cup \rightarrow^{ST1} \cup \rightarrow^{ST3} \cup \rightarrow^{Terminal} \cup \rightarrow^{ST2}$ immediately fire once they are enabled, their processing is assumed to take zero time and the new configuration $\mathcal{C}'$ differs from the preceding $\mathcal{C}$ only in switching from n_k to n_l. This kind of $\vdash$ transitions reflects the fact that $\rightarrow^{Start} \cup \rightarrow^{ST1} \cup \rightarrow^{ST3} \cup \rightarrow^{Terminal} \cup \rightarrow^{ST2}$ have been introduced for creating ebBP compliant models only and that there is no logic in empty BTAs. The remaining elements of $\vdash$ can be derived using the set of rules below that represent *triggering BTAs* ($\vdash^{\rightarrow^{Trigger}}$), *finishing BTAs* ($\vdash^{\rightarrow^{Eval}}, \vdash^{\rightarrow^{Update}}$), *evaluating BTAs* ($\vdash^{\rightarrow^{Route}}$), *leaving STs by timeout* ($\vdash^{\rightarrow^{Timeout}}$) and *the elapse of time* ($\vdash^{clock}$).

1: Trigger a BTA
$(n, o, v_t, v_{th}) \vdash^{\rightarrow^{Trigger}}$
$(n', o, v_t, v_{th}) \;\; iff$
$(n, g, n') \in \rightarrow^{Trigger} \wedge$
$\psi(g, (n, o, v_t, v_{th})) = true \wedge$
$((v_{th} = -1) \vee (v_t < v_{th}))$

2: Finish a BTA and start result evaluation
$(n, o, v_t, v_{th}) \vdash^{\rightarrow^{Eval}}$
$(n', o', v_t, v_{th}) \;\; iff$
$(n, g, n') \in \rightarrow^{Eval} \wedge$
$\psi(g, (n, o, v_t, v_{th})) = true \wedge$
$o' = \chi(n)$

3: Finish a BTA and ignore result
$(n, o, v_t, v_{th}) \vdash^{\rightarrow^{Update}}$
$(n', o, v'_t, v'_{th}) \;\; iff$
$(n, g, n') \in \rightarrow^{Update} \wedge$
$\psi(g, (n, o, v_t, v_{th})) = true \wedge$
$((n' = nodeb(btaSrc(n)) \wedge v'_t = 0 \wedge$
$v'_{th} = \phi(nodef(btaSrc(n)))) \vee$
$(n' = nodei(btaSrc(n)) \wedge v'_t = v_t \wedge v'_{th} = v'_{th}))$

```
4:   Evaluate a BTA result
     $(n, o, v_t, v_{th}) \vdash^{\rightarrow^{Route}}$
     $(n', o', v'_t, v'_{th})\ \ iff$
     $(n, g, n') \in \rightarrow^{Route} \wedge$
     $\psi(g, (n, o, v_t, v_{th})) = true \wedge$
     $o' = \{\} \wedge$
     $((parentST(n') = \bot \wedge v'_t = v_t \wedge v'_{th} = -1) \vee$
     $(parentST(n') \neq \bot \wedge n' = nodei(decSrc(n)) \wedge$
     $v'_t = v_t \wedge v'_{th} = v'_{th}) \vee$
     $(parentST(n') \neq \bot \wedge n' \neq nodei(decSrc(n)) \wedge$
     $v'_t = 0 \wedge v'_{th} = \phi(nodef(parentST(n')))))$
5:   Leave ST by timeout
     $(n, o, v_t, v_{th}) \vdash^{\rightarrow^{Timeout}}$
     $(n', o, v'_t, v'_{th})\ \ iff$
     $(n, g, n') \in \rightarrow^{Timeout} \wedge$
     $\psi(g, (n, o, v_t, v_{th})) = true \wedge$
     $((v_{th} > -1) \wedge (v_t >= v_{th})) \wedge$
     $((parentST(n') = \bot \wedge v'_t = v_t \wedge v'_{th} = -1) \vee$
     $((parentST(n') \neq \bot) \wedge v'_t = 0 \wedge$
     $v'_{th} = \phi(nodef(parentST(n')))))$
6:   Elapse of time
     $(n, o, v_t, v_{th}) \vdash^{clock}$
     $(n, o, v'_t, v_{th})\ \ iff$
     $v'_t = v_t + 1$
```

Finally, a WSTBC is defined to be valid if and only if every node n ∈ N is connected to the initial node and one terminal node and there exists a configuration such that the terminal node is reachable by a sequence of transition steps as defined above.

Definition 4.4.3 (Validity)
Let $\vdash^*$ be the transitive closure of $\vdash$. A WSTBC is valid iff

(i) Each $n \in N$ is connected *to both $\{s_0\}$ and at least one $t \in T$, and*

(ii) $\forall n \in N : \exists \mathcal{C} = (n, o, v_t, v_{th}) :$
$(n, o, v_t, v_{th}) \vdash^* (n', o', v'_t, v'_{th}) \wedge n' \in T$

4.4.3.3. ESTBC

The core idea of the extension-type STBC (ESTBC) model is replacing the 4 nodes that represent a shared state in the WSTBC model by a single special-purpose node.

This is a change at the syntactical level and instead of rephrasing large parts of the WSTBC definitions, the core differences between ESTBC and WSTBC models are presented. Language restrictions, facts and semantics then hold analogously.
The node set N of an ESTBC is defined as the union $\{s_0\} \cup \mathrm{ST} \cup \mathrm{DEC} \cup \mathrm{SBTA} \cup \mathrm{T}$ where ST denotes a set of shared states as introduced in section 4.4.2.1. Compared to the definition of WSTBC, the node sets FORK, JOIN and SEBTA are missing as these are needed only for representing a shared state in an ebBP compliant way. Conflating

Table 4.1.: WSTBC/ESTBC Transition Relations

WSTBC	ESTBC
$\rightarrow^{Start}$	$\rightarrow^{Start} \subseteq s_0 \times \{tt\} \times \mathrm{ST}$ and $\left\|\rightarrow^{Start}\right\| = 1$
$\rightarrow^{ST1}$	No correspondence
$\rightarrow^{ST3}$	No correspondence
$\rightarrow^{Terminal}$	via $\rightarrow^{Route'}$ and $\rightarrow^{Timeout'}$
$\rightarrow^{ST2}$	No correspondence
$\rightarrow^{Trigger}$	$\rightarrow^{Trigger} \subseteq \mathrm{ST} \times \{tt\} \times \mathrm{SBTA}$
$\rightarrow^{Eval}$	Identical
$\rightarrow^{Update}$	$\rightarrow^{Update} \subseteq \mathrm{SBTA} \times \{tt\} \times \mathrm{F} \times \mathrm{ST}$
$\rightarrow^{Route}$	$\rightarrow^{Route} \subseteq \mathrm{DEC} \times 2^G \setminus g^{to} \times \mathrm{F} \times \mathrm{ST}$
$\rightarrow^{Route}$	$\rightarrow^{Route'} \subseteq \mathrm{DEC} \times 2^G \setminus g^{to} \times \mathrm{T}$
$\rightarrow^{Timeout}$	$\rightarrow^{Timeout} \subseteq \mathrm{ST} \times g^{to} \times \mathrm{ST}$
$\rightarrow^{Timeout}$	$\rightarrow^{Timeout'} \subseteq \mathrm{ST} \times g^{to} \times \mathrm{T}$

the WSTBC-based shared state components to a single element also influences some other elements. In particular, ebBP's `toBusinessStateRef/fromBusinessStateRef` constraint [134, sec. 3.8.2] has been dropped in order to allow for linking to shared states. Consistently, empty BTAs are not needed for linking to final states (T) any more. Instead, terminal nodes are reached in ESTBCs directly from STs or `Decision` nodes. Moreover, as the reset of timeout values cannot be deduced any more from whether the first or third node of a shared state component is the target of a link, this information is explicitly encoded into the links. Some elements therefore have an additional flag $\mathrm{f} \in \mathrm{F} = \{tt, ff\}$ that indicates whether the targeted final state shall reset its timer or not. Table 4.1 compares the WSTBC and ESTBC transition relations where the left column names the WSTBC relations and the right column describes the corresponding ESTBC definition The relation names have not been changed for emphasizing the semantic similarity and 'g^{to}' is used as abbreviation

for $\{(\text{XPath1}, \mathit{timeout})\}$. Finally, the ϕ function that maps every shared state to its corresponding timeout value is now defined on $\text{ST} \rightarrow \mathbb{N}_0 \cup \{-1\}$ instead of $\text{FORK} \rightarrow \mathbb{N}_0 \cup \{-1\}$.
The translation of ESTBCs into WSTBCs is presented using a pseudo-algorithm (algorithm objects 1, 2 and 3). The basic idea of the algorithm is first translating the input ESTBC's shared states and terminal nodes and associating these with the *EmptyBTAs* that have been generated during translation. During translation of the transition relation elements, references to the input ESTBC's shared states and terminal nodes then are mapped correspondingly.

Algorithm 1: ESTBC to WSTBC Conversion: part 1

```
input      : A valid ESTBC $ebc$ to be transformed
output     : A valid WSTBC $wbc$
// map data structures for already mapped nodes
variables  : $stMap$<ST,SEBTA>; $tMap$<T,SEBTA>
algorithm :
// copy components that remain unchanged
$wbc$.R = $ebc$.R;
$wbc.s_0 = ebc.s_0$;
$wbc$.DEC = $ebc$.DEC;
$wbc$.SBTA = $ebc$.SBTA;
$wbc$.T = $ebc$.T;
$wbc$.G = $ebc$.G;
$wbc.\rightarrow^{Eval} = ebc.\rightarrow^{Eval}$;
// Create empty BTAs before terminal nodes
foreach $t$ in $ebc.T$ do
    $et$ = createEmptyNode();
    $tMap$.add($t$,$et$);
    $wbc.\rightarrow^{Terminal}$.add(($et$,{$tt$},$t$));
end
// Translate shared states
foreach $st$ in $ebc.ST$ do
    $e_b$ = createEmptyNode();
    $e_i$ = createEmptyNode();
    $j$ = createJoinNode();
    $f$ = createForkNode();
    $f$.th = $st$.th;
    $wbc$.SEBTA.add($e_b$);
    $wbc$.SEBTA.add($e_i$);
    $wbc$.JOIN.add($j$);
    $wbc$.FORK.add($f$);
    $wbc.\rightarrow^{ST1}$.add(($e_b$,{$tt$},$j$));
    $wbc.\rightarrow^{ST2}$.add(($j$,{$tt$},$e_i$));
    $wbc.\rightarrow^{ST3}$.add(($e_i$,{$tt$},$f$));
    $stMap$.add($st$,$eb$);
end
// continue...
```

Algorithm 2: ESTBC to WSTBC Conversion: part 2

```
// ...continue
// Translate Transitions
foreach $(n_1,g,n_2)$ in $ebc.\rightarrow^{Start}$ do
    $e_b = stMap$.get($n_2$);
    $wbc.\rightarrow^{Start}$.add($(n_1,g,e_b)$);
end
foreach $(n_1,g,n_2)$ in $ebc.\rightarrow^{Trigger}$ do
    $f$ = nodef(parentST($stMap$.get($n_1$)));
    $wbc.\rightarrow^{Trigger}$.add($(f,g,n_2)$);
end
foreach $(n_1,g,r,n_2)$ in $ebc.\rightarrow^{Update}$ do
    if $r == true$ then
        $e$ = nodeb(parentST($stMap$.get($n_2$)));
    else
        $e$ = nodei(parentST($stMap$.get($n_2$)));
    end
    $wbc.\rightarrow^{Update}$.add($(n_1,g,e)$);
end
foreach $(n_1,g,r,n_2)$ in $ebc.\rightarrow^{Route}$ do
    if $r == true$ then
        $e$ = nodeb(parentST($stMap$.get($n_2$)));
    else
        $e$ = nodei(parentST($stMap$.get($n_2$)));
    end
    $wbc.\rightarrow^{Route}$.add($(n_1,g,e)$);
end
// ...continue
```

Algorithm 3: ESTBC to WSTBC Conversion: part 3

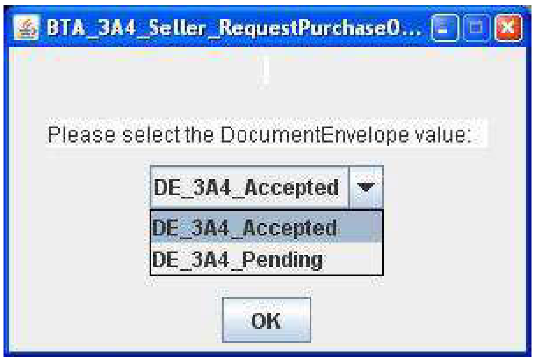
```
// ...continue
foreach $(n_1,g,n_2)$ in $ebc.\rightarrow^{Route'}$ do
    $e$ = $tMap$.get($n_2$);
    $wbc.\rightarrow^{Route}$.add($(n_1,g,e)$);
end
foreach $(n_1,g,n_2)$ in $ebc.\rightarrow^{Timeout}$ do
    $f$ = nodef(parentST($stMap$.get($n_1$)));
    $e$ = nodeb(parentST($stMap$.get($n_2$)));
    $wbc.\rightarrow^{Timeout}$.add($(f,g,e)$);
end
foreach $(n_1,g,n_2)$ in $ebc.\rightarrow^{Timeout'}$ do
    $f$ = nodef(parentST($stMap$.get($n_1$)));
    $e$ = $tMap$.get($n_2$);
    $wbc.\rightarrow^{Timeout}$.add($(f,g,e)$);
end
return $wbc$;
```

4.4.4. Evaluation

The evaluation of this work concerns three main areas. Most important is the practical feasibility of the ebBP-2-BPEL translation algorithm assuming the integration architecture introduced in section 4.2. Furthermore, the computational complexity of the ESTBC-WSTBC conversion and ebBP-2-BPEL translation algorithms is analyzed. Finally, the possible reduction of extensional complexity using ESTBCs instead of WSTBCs is discussed. Note again that the implementation of the ebBP-2-BPEL translation algorithm stems from Christoph Pflügler's diploma thesis and therefore just a high-level overview of the implementation is given. Details are available in [160, 182] and the algorithm itself is given in appendix B.

For evaluating the feasibility of the proposed ST-based ebBP-BPEL translation approach a translator has been written in the Java language; the main API used for that was the Streaming API for XML (StAX (cf. `http://jcp.org/en/jsr/detail?id=173`, last access: 12/20/2011)). StAX does not require to load the complete XML document to be processed into memory and therefore it is theoretically possible to process arbitrarily large ebBP choreographies. In practice, this is only helpful in case the generated BPEL processes can still be executed on BPEL engines. On the other hand, the application of StAX is more laborious than using DOM based APIs like JAXB (cf. `http://jcp.org/en/jsr/detail?id=222`, last access: 12/20/2011). Approximately 14000 method lines of code have been written to implement the translator. Less code may have been needed using other libraries like DOM or other technologies like XSLT. For the approach presented here, the choice of technology for implementing the translator is of minor importance.
As regards the feasibility of translating arbitrary valid WSTBCs, please note that directed graphs without concurrency basically are state machines with multiple types of nodes. A valid WSTBC only contains `Fork` nodes of type 'XOR' and therefore is a directed graph without concurrency. The implementing BPEL processes use a global while loop for switching across a WSTBC's STs using a global variable for determining the current ST. Once a timeout occurs or a BTA has been performed that results in a ST change, this global variable is assigned correspondingly and the new ST is reached in the next iteration of the global while loop. Such a BPEL process essentially amounts to the implementation of a state machine. The use case of section 4.4.1 can be translated in full and produces fully BPEL compliant process descriptions. Translating the use case takes approximately 45 seconds using a Centrino duo 1,8 GHz machine with 2 GB RAM. Note that the produced BPEL processes are immediately ready for deployment as the incorporation of business logic is allowed for by predefined interfaces for accessing backend systems. The BPEL processes created have been tested using the Apache ODE 1.2 BPEL engine and the Apache Axis2 1.4 Web Services stack. For the backend services described above, dummy Web services have been implemented that emulate business logic by forwarding decisions to the user. Figure 4.9 shows the *Seller* role deciding whether to accept an order in full (Accepted) or to defer its decision (Pending). Once the user selects the *DE_3A4_Accepted* option and the BT protocol terminates successfully,

the ST *Contract* is reached which is displayed in figure 4.10. The use case from



Please select the DocumentEnvelope value:

Figure 4.9.: Seller Deciding upon Quote Request

State Contract reached.

Figure 4.10.: Seller Entering State *Contract*

section 4.4.1 could not be performed on the selected platform in full due to an ODE bug in handling `whiles` in combination with `picks` which resulted in the situation that a shared state's `while` element can only be entered once. Thus, though every shared state of the use case could be reached there were two states that could not be followed-on with a "normal" termination of the process.
In practice, an important question is the validity of the defined assumption that performing a BTA *"either leads to an agreed-upon result or a protocol failure"* defined in section 4.4.3. This is problematic because integration partners are assumed to perform BTAs collaboratively using separate BPEL processes. This introduces the problem of communication over unreliable media. Even worse, BTAs may require the realization of several B2Bi-related security features. Note that this topic conceptually has been discussed in section 4.3.

For the platform selected for ebBP-ST implementation, the use of WS-ReliableMessaging (using Apache Sandesha2, cf. `http://ws.apache.org/sandesha/sandesha2/`, last access: 12/20/2011) and WS-Security (using Apache Rampart, cf. `http://ws.apache.org/rampart/`, last access: 12/20/2011) has been considered for implementing QoS features. Though it was possible to offer BPEL processes as secure and reliable Web Services, invoking other Web Services from BPEL processes in a reliable and secure manner was not possible. So the application of these standards has been canceled for the ebBP-ST target platform. Chapter 5 and appendix C discuss the realization of reliability and security using the WS-* implementations of GlassFish and IBM WebSphere Application Server. As a result, the assumption of mutually agreed upon BTA results using Web Services technology can be assumed to be realistic for selected homogeneous environments.

The main algorithm proposed for ebBP-ST is the ESTBC-WSTBC (section 4.4.3.3) conversion algorithm. As regards the ESTBC-WSTBC algorithm, the computational complexity is fairly low. If a hash-map data structure which provides linear time access can be used and the number of nodes of an ESTBC is n then the complexity of translating an ESTBC into a WSTBC is linear, i.e., $\mathcal{O}(\mathrm{n})$. In practice the complexity of parsing an input XML file and writing an output XML file must be considered as well. The first one can easily be done using standard tools. The second one is trivial as well because the ebBP schema does not impose any restrictions on the ordering of `Forks`, `Joins`, `BTAs`, `Decisions` and terminal nodes.

The ebBP-BPEL translation algorithm (see appendix B) developed by Christoph Pflügler also scales well. As `Joins` and `Forks` are used for the representation of shared states only, the identification of shared states takes at most one iteration across a WSTBC's nodes. For creating each resulting BPEL's global while loop (lines 2-13 of algorithm 11 on page 302) each ST is then visited once. Afterwards, the placeholders for BTAs and the corresponding *Decisions* can be processed in the order they have been written to the BPEL output processes and looking up the matching BTAs and *Decisions* of the input WSTBC takes linear time as well if these have been stored in the first iteration in a hashmap-like data structure. This means that this algorithm also has linear time complexity $\mathcal{O}(\mathrm{n})$ in terms of the number of nodes n.
Finally, the complexity reduction achievable by using the ebBP-ST schema extension is assessed. Replacing the ebBP schema compliant representation of a ST with the more compact extension-based model as described in section 4.4.2.2 obviously leads to much more compact XML code. Table 4.2 summarizes the reduction of extensional complexity for this section's use case. The lines of code (LOC) metric refers to the representation of the ebBP `BusinessCollaboration` element and disregards the declaration of DocumentEnvelope definitions as well as BusinessTransaction type definitions that is the same for both types of ST representation. A more valid approach to measure complexity that is not as dependent on formatting is counting the ebBP elements needed for representing nodes and transitions. The first column of table 4.2 names the *complexity metric/ebBP element* considered and the other columns contain the measured values for this section's use case as well as the ratio of reduction. Note that the element reduction ratio depends on the collaboration's process structure. Considering the selected elements above, the node reduction ratio can exactly be calculated in terms of node sets via
$(3*|SH_{WSTBC}| + |\mathrm{T}|) / (|\mathrm{SBTA}| + |\mathrm{SEBTA}| + 2*|SH_{WSTBC}| + |\mathrm{T}| + |\mathrm{DEC}|)$. Obviously, the reduction ratio depends on the ratio of |SBTA| to $|SH_{WSTBC}|$. In the worst case, if there is only one ST component and a very high number of BTAs that

Table 4.2.: WSTBC/ESTBC Complexity Comparison (Use Case)

Metric	WSTBC	ESTBC	Reduction
LOC	787	591	$\sim$ 0.249
BTA	31	15	$\sim$ 0.516
Fork	7	0	1
Join	7	0	1
Decision	13	13	0
Success	1	1	0
Failure	1	1	0
ST	0	7	undefined
Sum of nodes	60	37	$\sim$ 0.383
ToLink	67	58	$\sim$ 0.134
FromLink	38	15	$\sim$ 0.605
Sum of links	105	73	$\sim$ 0.305

all are admissible for that ST and either link back to that ST or to a terminal node then the reduction ratio tends to 0. In the best case, if there is only one BTA for some initialization and a very high number of STs that are reachable via timeouts then the reduction ratio tends to (3*|SH_{WSTBC}|) / (|SEBTA| + 2*|SH_{WSTBC}|). The reduction ratio of `ToLinks`/`FromLinks` cannot exactly be calculated in terms of node sets because `Decisions` may have a varying number of branches and a ST may or may not have a timeout configured. Moreover, for STs without timeout and with only one admissible BTA the `ToLink` to the following BTA must be configured twice to conform with ebBP schema restrictions. Removable links can be calculated as (4*|SH_{WSTBC}| + |T|). For the calculation of the overall number of links assume that every ST has a timeout configured and that each `Decision` has three branches. Due to ebBP's toBusinessStateRef/fromBusinessStateRef restriction every ToLink/FromLink must reference a BTA for WSTBCs. The number of FromLinks then is (|SBTA| + |SEBTA| + |SH_{WSTBC}|) because in every ST the first component is connected twice. The number of ToLinks can be derived by considering the different types of elements that link to BTAs or EmptyBTAs, i.e., (1 + |SH_{WSTBC}| + |JOIN| + |SBTA| + 3*|DEC| + (|BTA|-|DEC|)). The number of ToLinks in `Fork` elements is captured by |SBTA| and (|SBTA|-|DEC|) captures the number of BTAs that directly link back to the source ST. Considering the relation between STs and its components, the link reduction ratio can be calculated as (4*|SH_{WSTBC}| + |T|) / (1 + 3*|SBTA| + 5*|SH_{WSTBC}| + |T| + 2*|DEC|). The best case/worst case analysis then is similar to the analysis for the node reduction ratio.

This section has shown how to represent shared states in binary ebBP choreography models using either an ebBP extension or an ebBP compliant workaround. ebBP-ST has been introduced as choreography style for capturing such models and a formal model for ebBP-ST has been given. Restrictions have been defined for characterizing valid ebBP-ST models and a formal operational execution semantics for valid ebBP-ST models has been given. In addition, a translation algorithm from extension-based ebBP-ST models to workaround-based ebBP-ST models has been defined. The implementability of the ebBP-ST execution semantics has been demonstrated in [160, 182] and will not be discussed in full here. The corresponding *ebBP-ST to BPEL* translation algorithm developed by Christoph Pflügler is given in appendix B.

The next section will focus on ebBP-Reg that does not explicitly offer the concept of *shared states*. However, ebBP-Reg models do neither rely on an ebBP extension nor on a corresponding workaround. Moreover, ebBP-Reg offers parallel composition and hierarchical decomposition.

4.5. ebBP-Reg Choreographies

In chapter 3, regular ebBP choreographies (ebBP-Reg, published in [190]) have been proposed as the ebBP choreography style that strictly conforms to the ebBP XML schema and supports hierarchical decomposition as well as parallelism for binary collaborations. Conceptually, ebBP-Reg is a variation of ebBP-ST that trades simplicity for control flow expressiveness. However, the concept of allowing for almost arbitrary control flow graphs in order not to impose too tight restrictions upon modelers is retained.

The most important differences between ebBP-Reg and ebBP-ST concern explicit shared states, hierarchical decomposition and parallelism. Explicit modeling of shared states is given up in order to ensure strict standard compliance. Parallelism and hierarchical decomposition are introduced in order to enhance control flow expressiveness. Hierarchical decomposition is achieved by means of allowing for BCAs at (almost) the same control flow points as BTAs. Parallel structures are represented in a special way in order to allow for intuitive control flow definition that may include irreducible loops. Basically, a parallel structure is defined to be a state that consists of one or more non-overlapping control flow branches. In order to keep up the state paradigm, all branches must be activated together in order to be executed and all branches must have terminated before control flow may leave the parallel structure. From a modeler's perspective, this can easily be facilitated by representing each branch as a separate BCA.

The presentation of ebBP-Reg is split up as follows. Section 4.5.1 introduces a use case that covers all control flow elements of ebBP-Reg to demonstrate its expressiveness. Section 4.5.2 then presents the formalization of ebBP-Reg as well as a characterization of validity. The execution semantics of ebBP-Reg is described in section 4.5.3. The validation of ebBP-Reg has been performed by means of creating a BPEL mapping for the ebBP-Reg language elements and a prototype that implements the use case of section 4.5.1. Both is described in section 5.3 so that a dedicated evaluation sub-section is left out here.

4.5.1. Use Case

For giving an impression of ebBP-Reg's expressiveness and for validation purposes, the artificial RosettaNet PIPs (conceptually equivalent to BTs) based purchasing use case depicted in figure 4.11 is used. The use case starts out with a parallel structure for concurrently exchanging purchase order requests (PIP 3A19) within two BCAs of the same type (BC-single3A19-1, BC-single3A19-2) that, again, specify seller and buyer roles. For BC-single3A19-1, the root level seller role takes the BC-single3A19 seller role and for BC-single3A19-2, the root level seller role takes the BC-single3A19 buyer role. After the concurrent purchase order requests, a single purchase order confirmation (PIP 3A20, denoted 'BT-3A20') is exchanged using a BTA. In case of a protocol failure (denoted 'P-F') or a negative confirmation message (denoted 'Stop'), the process is terminated. Otherwise, the process is continued and multiple actions may be taken. The root level buyer role may try to change (PIP 3A21) or cancel (PIP 3A23) the purchasing process or the root level seller role may try to finish the process by sending an invoice (PIP 3C3). Depending on whether or not these BTAs succeed ('[P-S]' transitions) or fail ('[P-F]' transitions) and depending on the result of additional BTAs (BT-3A22/3A24 for replying to change/cancellation requests) the process eventually terminates. The different final states denote the different overall results of the purchasing process.

It is worth noting that the use case captures the main types of process components (parallel structures, loops, event-based choices) that can be created from the rules of the next section. This use case has also been used to validate the mapping of ebBP-Reg to BPEL as described in section 5.3.

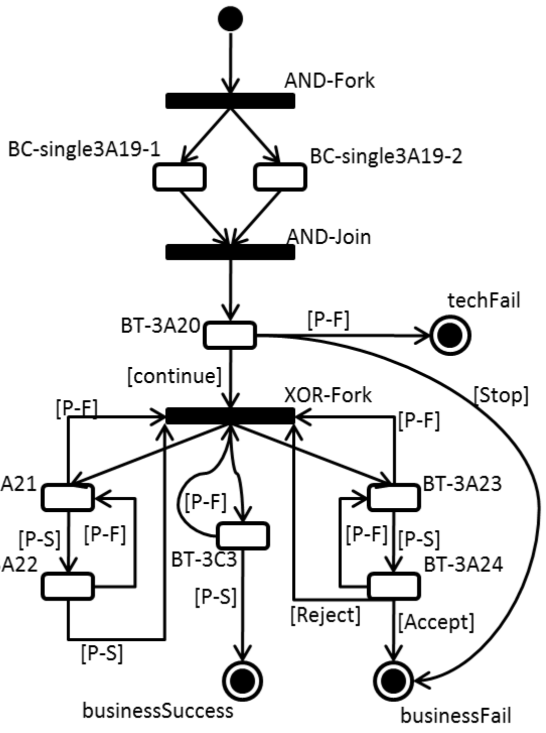


Figure 4.11.: ebBP-Reg Use Case ([true] guards left out)

4.5.2. Formalization of ebBP-Reg

First, a superset of the eligible ebBP-Reg processes is defined that is restricted by means of wellformedness rules afterwards.

Definition 4.5.1 (ebBP-Reg Process)
An ebBP-Reg process is a five-tuple RP (s_0, F,A,C,T) with the following elements:

- s_0 *the start node.*
- *F a non-empty set of final states.*

- *A = SBTA ∪ SBCA with SBTA a non-empty set of BTAs and SBCA a set of BCAs.*
- *C = SXORF ∪ SANDF ∪ SANDJ with SXORF a set of XOR-Forks, SANDF a set of AND-Forks and SANDJ a set of AND-Joins.*
- *T the union of the following transition sets*
  - $T^{start} = \{(s_0, true, e)\}$, $e \in (A \cup SXORF \cup SANDF)$
  - $T^{end} = A \times G \times F$
  - $T^{ctrlIn} = A \times G \times C$
  - $T^{ctrlOut} = C \times \{true\} \times A$
  - $T^{straight} = A \times G \times A$

  where $G = G^{bta} \cup G^{bca}$ *a set of boolean guards defined on the results of BTAs and BCAs, respectively.* □

Furthermore, the following auxiliary functions are defined.

Definition 4.5.2 (Auxiliary Functions)

- *.-notation/#-notation is used for accessing the components of a tuple by name/index.*
- *A path between two nodes* $a, b \in \{s_0\} \cup A \cup C \cup F$ *is a sequence of nodes* $a, n_{1..x}, b$ *such that for all i=1...x-1,* (n_i, g_i, n_{i+1}) *and* (a, g_a, n_1) *and* $(n_x, g_x, b) \in T$. *Let Path(a,b) be the set of all paths between a and b.* □

Considering the ebBP control flow constructs introduced in section 2.3.1, OR-Forks, OR-Joins and Decisions are missing in the definition of ebBP-Reg processes. The functionality of OR-Forks to perform an arbitrary selection of follow-on activities is not supported. Consequently, OR-Joins are not needed because any node n ∈ A ∪ C \ SANDJ then can serve as join node for an XOR-Fork and AND-Joins can be used for joining AND-Forks. The functionality of ebBP Decisions is provided using the guards on ebBP Transitions in order to circumvent referential constraint problems when linking from Decisions to other control flow nodes (cf. toBusinessStateRef/fromBusinessStateRef constraints in the ebBP standard [134, section 3.8.2]).
The following production rules describe how to create a syntactically valid ebBP-Reg model step by step. The production rules are chosen such that any syntactically valid ebBP-Reg model can be performed using BPEL (cf. section 5.3) later on.

The first rule characterizes an elementary valid ebBP-Reg process that consists of a single activity.

Rule 4.5.1 (Elementary Process 1)
Any process $rp = (s_0, \{f\}, \{a\}, \emptyset, \{(s_0, true, a), (a, true, f)\})$ *is a valid RP.* □

The second rule characterizes an elementary valid ebBP-Reg process that starts with a so-called *event-based choice* that selects from a series of activities. Each activity is followed by a separate final state.

Rule 4.5.2 (Elementary Process 2)
Let ebc = (xorF, F, A, $T^{ctrlOut}$, T^{end}) be an 'event-based choice component' that chooses from a set of BTAs/BCAs at run-time as requested by a backend/partner process with:

- *xorF an XOR-Fork.*
- *$F = \{f_1,...,f_n\}$ a set of final states.*
- *$A = \{a_1,...,a_n\}$ a set of BTAs and BCAs with at least one BTA.*
- *$T^{ctrlOut} = xorF \times \{true\} \times A$.*
- *$T^{end} \subseteq A \times \{true\} \times F$ such that $A \times \{true\} \rightarrow F$ is a bijective function.*

Then, rp = (s_0, ebc.F, ebc.A, {ebc.xorF}, {(s_0, true, ebc.xorF)} ∪ ebc.$T^{ctrlOut}$ ∪ ebc.T^{end}) is a valid RP. □

The third rule represents an elementary valid ebBP-Reg process that starts with a parallel structure. The process terminates in a final state after the parallel structure.

Rule 4.5.3 (Elementary Process 3)
Let $rp_1,...,rp_n$ be a set of valid RPs and PCA = $bca^{rp_1},...,bca^{rp_n}$ be a set of BCAs for executing each RP. Let andF, andJ be an AND-Fork and an AND-Join and let $T^{fork} = \{andF\} \times \{true\} \times PCA$ and $T^{join} = PCA \times \{true\} \times \{andJ\}$.
Then, rp = (s_0, {f}, PCA, {andF,andJ}, {(s_0, true, andF),(andJ, true, f)} ∪ T^{fork} ∪ T^{join}) is a valid RP. □

The fourth rule shows how to add a BTA to the end of a valid ebBP-Reg process, that means immediately before an existing final state.

Rule 4.5.4 (Add BTA)
Let rp be a valid RP, bta a BTA ∉ rp.A.SBTA, and pred = (pred#1,pred#2,pred#3) ∈ rp.T^{end}.
Then rp'= (rp.s_0, rp.F, rp.A.SBTA ∪ {bta}, rp.C, rp.T ∪ {(pred#1, pred#2, bta), (bta, true, pred#3)} \ {(pred#1, pred#2, pred#3)}) is a valid RP. □

The fifth rule shows how to add a sub-process to the end of a valid ebBP-Reg process.

Rule 4.5.5 (Process Composition)
Let rp_1, rp_2 be valid RPs, $pred \in rp_1.T^{end}$, bca^{rp_2} be a BCA executing rp_2, and $bca^{rp_2} \notin rp_1.A.SBCA$.
Then rp' = $(rp_1.s_0, rp_1.F, rp_1.A.SBCA \cup \{bca^{rp_2}\}, rp_1.C, rp_1.T \cup \{(pred\#1, pred\#2, bca^{rp_2}), (bca^{rp_2}, true, pred\#3)\} \setminus \{(pred\#1, pred\#2, pred\#3)\})$ is a valid RP. $\square$

The sixth rule shows how to add an event-based choice together with a series of activities for each branch of the event-based choice to the end of a valid ebBP-Reg process.

Rule 4.5.6 (Event-Based Choice)
Let rp be a valid RP and $pred \in rp.T^{end}$. Let ebc = $(xorF, F, A, T^{ctrlOut}, T^{end})$ be an 'event-based choice component' that chooses from a set of BTAs/BCAs at run-time as requested by a backend/partner process with:

- *xorF an XOR-Fork.*
- *$F = \{f_1,...,f_n\}$ a set of final states.*
- *$A = \{a_1,...,a_n\}$ a set of BTAs and BCAs with at least one BTA.*
- *$T^{ctrlOut} = xorF \times \{true\} \times A$.*
- *$T^{end} \subseteq A \times \{true\} \times F$ such that $A \times \{true\} \rightarrow F$ is a bijective function.*

Furthermore, let $ebc.xorF \notin rp.C \wedge (rp.F \setminus \{pred\#3\}) \cap ebc.F = \emptyset \wedge ebc.A \cap rp.A = \emptyset$.
Then rp' = $(rp.s_0, (rp.F \setminus \{pred\#3\}) \cup ebc.F, rp.A \cup ebc.A, rp.C \cup \{xorF\}, rp.T \cup ebc.T^{ctrlOut} \cup ebc.T^{end} \cup \{(pred\#1,pred\#2,xorF)\} \setminus \{(pred\#1, pred\#2, pred\#3)\})$ is a valid RP. $\square$

The seventh rule shows how to add a parallel structure to the end of a valid ebBP-Reg process.

Rule 4.5.7 (Parallel)
Let rp_x be a valid RP and $pred \in rp_x.T^{end}$. Let $rp_1,...,rp_n$ be a set of valid RPs and $PCA = bca^{rp_1},...,bca^{rp_n}$ be a set of BCAs for executing each RP and for all bca^{rp_i} with i in [1;n] holds: $bca^{rp_i} \notin rp_x.A$. Let andF, andJ $\notin rp_x.C$ be an AND-Fork and an AND-Join and let $T^{fork} = \{andF\} \times \{true\} \times PCA$ and $T^{join} = PCA \times \{true\} \times \{andJ\}$.

Let for any rp_i,rp_j with i,j in [1;n], $i \neq j$: All constituent sets of rp_i and rp_x as well as rp_i and rp_j are pairwise disjoint.
Then, rp' = $(rp_x.s_0, rp_x.F, rp_x.SBCA \cup PCA, rp_x.C \cup \{andF, andJ\}, rp_x.T \cup T^{fork} \cup T^{join} \cup \{(pred\#1,pred\#2,andF), (andJ,true,pred\#3)\} \setminus \{(pred\#1, pred\#2, pred\#3)\})$ is a valid RP. $\square$

For the following rules, let rp.A.PCA denote the set of BCAs within any parallel structure of a given valid RP rp as defined in rule 4.5.7.
The ninth rule shows how to add a transition between some activity (outside of any parallel structure) of a valid ebBP-Reg process and any other *state* of the ebBP-Reg process. In that regard, it is vital to note that a parallel structure is interpreted as a state of the surrounding process.

Rule 4.5.8 (Add Transition)
Let rp be a valid RP, $d \in (rp.A \setminus rp.A.PCA) \cup rp.C.SXORF \cup rp.C.SANDF \cup rp.F$, $p^{add} \in rp.A \setminus rp.A.PCA$, and $T_{p^{add}} = \{t \in rp.T \mid t\#1 = p^{add}\}$.

Furthermore, let newT = (p^{add},newG,d) be a transition, R be a function that computes a new set of transitions from $T_{p^{add}}$ by assigning a new valid guard to each element of $T_{p^{add}}$ such that (newG $\vee \bigvee_{t \in R(T_{p^{add}})}$ t#2) evaluates to true and
$\forall$ t1,t2 $\in$ newT $\cup R(T_{p^{add}})$, t1 $\neq$ t2: (t1#2 $\wedge$ t2#2) evaluates to false.
Then, ($rp.s_0$, rp.F, rp.A, rp.C, $(T \setminus T_{p^{add}}) \cup \{newT\} \cup R(T_{p^{add}})$) is a valid RP. □
Note that this rule also is valid for d $\in$ rp.F and therefore also covers new transitions to final nodes.

The tenth rule shows how to add a final node immediately after some activity of a valid ebBP-Reg process where the respective activity must not be contained within any parallel structure.

Rule 4.5.9 (Add Final Node)
Let rp be a valid RP, $p^{add} \in rp.A \setminus rp.A.PCA$, and $T_{p^{add}} = \{t \in rp.T \mid t\#1 = p^{add}\}$.

Furthermore, let newF $\notin$ rp.F be a final node, newT = (p^{add},newG,newF) be a transition, R be a function that computes a new set of transitions from $T_{p^{add}}$ by assigning a new valid guard to each element of $T_{p^{add}}$ such that (newG $\vee \bigvee_{t \in R(T_{p^{add}})}$ t#2) evaluates to true, and $\forall$ t1,t2 $\in$ newT $\cup R(T_{p^{add}})$, t1 $\neq$ t2: (t1#2 $\wedge$ t2#2) evaluates to false.
Then, rp' = ($rp.s_0$, rp.F $\cup$ newF, rp.A, rp.C, $(T \cup \{newT\} \setminus T_{p^{add}}) \cup R(T_{p^{add}})$) is a valid RP. □

The eleventh rule shows how to remove a final node from a valid ebBP-Reg process. This rule is needed in order to allow the branches of an event-based choice to terminate in the same final state.

Rule 4.5.10 (Remove Final Node)
Let rp be a valid RP, f $\in$ rp.F, $PRED_f = \{t \in rp.T^{end} \mid t\#3 = f\}$, and H_f be the set of sets of outgoing transitions of predecessors of f such that: $PRED_f \subseteq \bigcup_{h \in H_f} h$ $\wedge \forall h_{x,f} \in H_f$: ($\forall t \in h_{x,f}$: t#1 = x) $\wedge$ ($\nexists t \in rp.T \setminus h_{x,f}$: t#1 = x) $\wedge$ $h_{x,f} \cap PRED_f$ $\neq \emptyset$.

Furthermore, let DEST $\subseteq$ *(rp.A* $\setminus$ *rp.A.PCA)* $\cup$ *rp.C.SXORF* $\cup$ *rp.C.SANDF* $\cup$ *rp.F* $\setminus$ *{f}.*

Furthermore, let R be a function that computes a new set of transitions from each $h_{x,f} \in H_f$ *by assigning to each* $t \in h_{x,f}$ *a new valid guard and replacing t#3 with some* $d \in$ *DEST such that:* $\bigvee_{t \in R(h_{x,f})}$ *t#2 evaluates to true* $\wedge$ $\forall$ *t1,t2* $\in R(h_{x,f})$*, t1* $\neq$ *t2: (t1#2* $\wedge$ *t2#2) evaluates to false, and for each* $h_{x,f} \in H_f$*:* $\exists$ *Path(x,e)* $\neq \emptyset$ *with e* $\in$ *rp.F* $\setminus$ *{f}.*
Then, rp' = (rp.s_0*, rp.F* $\setminus$ *{f}, rp.A, rp.C, (T* $\setminus \bigcup_{h \in H_f} h) \cup \bigcup_{h \in H_f} R(h))$ *is a valid RP.* □

4.5.3. ebBP-Reg Semantics

The paradigm for ebBP-Reg modeling is a state-machine that can hierarchically be decomposed and that allows for parallelism by means of structured AND-Fork/AND-Join combinations the branches of which are executed in a threading-like manner. ebBP-Reg can be interpreted as an evolution of ebBP-ST. Both B2Bi choreography styles are state-machine based and ebBP-Reg's XOR-Forks are very similar to shared states from a control flow perspective (except for timeouts). Yet, there are considerable syntactic and conceptual differences. The concept of shared states has been given up in ebBP-Reg so that several components of the ebBP-ST transition relation can be ignored. Conversely, the processing of parallel structures cannot directly be derived from the ebBP-ST semantics. Therefore, a new operational execution semantics for ebBP-Reg is given instead of describing how to derive it from the ebBP-ST semantics.

The execution semantics of ebBP-Reg defines the control flow of BCAs. For BTAs (similar to the ebBP-ST semantics), the assumption is made that performing a BTA takes some time and eventually either leads to an agreed-upon result or a protocol failure. The actual execution model for BTAs is described in section 4.3.

Let $\mathcal{C} \in EN \times EX \times R$ be the configuration of an ebBP-Reg choreography where N=$A \cup C \cup F$ is the set of choreography nodes as defined in definition 4.5.1, EN=2^N is the set of sets of currently enabled nodes, EX=2^N is the set of sets of currently active nodes, and R=$2^{A \times O}$ is the set of sets of nodes paired with the result of their latest execution. O is the set of all possible outcomes of all BTAs and BCAs of an ebBP-Reg choreography. Let χ be the function that computes the outcome of a BTA or BCA that has just finished. Furthermore, $\psi : G \times R \to \{true, false\}$ is the function that evaluates a given guard under a given set of activity results to one of the boolean constants *true* or *false*. In particular, $\psi(true, r) = true$ for arbitrary r $\in$ R. Any node n $\in$ N is *enabled* for some configuration $\mathcal{C} = (en, ex, r)$ if n $\in$ en.

The execution semantics of an ebBP-Reg process is defined operationally by the relation $\vdash \subseteq \mathcal{C} \times \mathcal{C}$. The initial configuration of an ebBP-Reg process is $(s_0, \emptyset, \emptyset)$ and immediately transitions to $(e, \emptyset, \emptyset)$ following the definition of the start transition T^{start} = $\{(s_0, true, e)\}$. The execution of an ebBP-Reg process terminates upon reaching a final configuration $(n, \emptyset, r)$ with n $\in$ F. The remaining elements of $\vdash$ are defined by the set of rules below that represent *starting an activity* ($\vdash^{act^{Start}}$), *finishing an activity* ($\vdash^{act^{Term}}$), *leaving an XOR-Fork or AND-Join* ($\vdash^{xor^{Out}}$), *processing an AND-Fork* ($\vdash^{par^{Start}}$), *activating a branch of a parallel structure* ($\vdash^{par^{Branch}}$) or *terminating a branch of a parallel structure* ($\vdash^{par^{Term}}$).

For the definition of the execution rules, par^1 and par^2 are defined as auxiliary functions such that:
par^1: $SANDF \rightarrow SBCA$ such that par^1(andf) $= \left\{bca|(andf, g, bca) \in T^{ctrlOut}\right\}$ for any andf $\in$ SANDF.
par^2: $SANDJ \rightarrow SBCA$ such that par^2(andj) $= \left\{bca|(bca, g, andj) \in T^{ctrlIn}\right\}$ for any andj $\in$ SANDJ.

1: Start an activity
$(en, ex, r) \vdash^{act^{Start}}$
$(en', ex', r')\ \ iff$
$en = \{n\} \wedge n \in A \wedge ex = \emptyset \wedge$
$en' = \emptyset \wedge ex' = \{n\} \wedge r' = r \wedge$
$(\nexists\ andF \in SANDF.\ n \in par^1(andF))$

2: Finish an activity
$(en, ex, r) \vdash^{act^{Term}}$
$(en', ex', r')\ \ iff$
$en = \emptyset \wedge ex = \{n\} \wedge n \in A \wedge$
$en' = \{n'\} \wedge r' = \{(n, \chi(n))\} \wedge$
$\exists (n, g, n') \in T^{end} \cup T^{ctrlIn} \cup T^{straight}.\psi(g, \chi(n)) = true$

3: Leave XOR-Fork or AND-Join
$(en, ex, r) \vdash^{xor^{Out}}$
$(en', ex', r')\ \ iff$
$en = \{n\} \wedge n \in SXORF \cup SANDJ \wedge ex = \emptyset \wedge$
$en' = \{n'\} \wedge ex' = \emptyset \wedge r' = \emptyset \wedge$
$\exists (n, true, n') \in T^{ctrlOut}$

4: Process AND-Fork
$(en, ex, r) \vdash^{par^{Start}}$
(en', ex', r') iff
$en = \{andF\} \wedge andF \in SANDF \wedge ex = \emptyset \wedge$
$en' = par^1(andF) \wedge ex' = \emptyset \wedge r' = \emptyset$

5: Activate branch of parallel structure
$(en, ex, r) \vdash^{par^{Branch}}$
(en', ex', r') iff
$en' = en \setminus \{n\} \wedge ex' = ex \cup \{n\} \wedge r' = r \wedge$
$\exists(andF, true, n) \in T^{ctrlOut}.\ andF \in SANDF \wedge n \in en$

6: Terminate branch of parallel structure
$(en, ex, r) \vdash^{par^{Term}}$
(en', ex', r') iff
$\exists(n, true, andj) \in T^{ctrlIn}.\ andj \in SANDJ \wedge n \in ex \wedge$
$(((en \neq \emptyset \vee ex \neq \{n\}) \wedge en' = en \wedge ex' = ex \setminus \{n\}) \vee$
$(en = \emptyset \wedge ex = \{n\} \wedge en' = \{andj\} \wedge ex' = \emptyset)) \wedge$
$r' = r \cup \{(n, \chi(n))\}$

The interested reader will have noticed that there is no rule for decomposition. Indeed, the execution semantics defined does not unfold hierarchies of BCAs. However, as each BCA must be performed independently of other BCAs, the semantics can simply be reapplied to each BCA of an ebBP-Reg process.

Furthermore, comparing the execution semantics to the definition of ebBP-Reg reveals that it is actually not strictly necessary to define both, $\vdash^{act^{Start}}$ and $\vdash^{act^{Term}}$, because no other rule can be fired when an activity is in execution. Similarly, keeping track of the result values of activities as part of $\mathcal{C}$ is not strictly necessary as this component of the configuration is not used for deciding upon firing rules. Both *add-ons* deliberately have been chosen. The two distinct rules for performing activities outside of parallel structures explicitly model that activity execution takes time. Hence, if timeout processing is needed in the future, the semantics will not prevent it. By analogy with the ebBP-ST definition, adding timeouts to XOR-Forks or BCAs is conceivable. Similarly, the explicit calculation of result values allows for adding expression based routing where it has not been defined so far. As the visualization of section 6.2 will reveal, this may be desirable for routing after AND-Joins.

This section has introduced ebBP-Reg as binary choreography style that can be expressed as strictly compliant ebBP. In particular, a formalization of ebBP-Reg has been defined and the validity of ebBP-Reg models has been characterized. Moreover, a formal execution semantics of ebBP-Reg has been defined the implementability of which based on BPEL processes will be shown in chapter 5. Before that, the next section will introduce SeqMP as multi-party choreography style.

4.6. SeqMP Choreographies

SeqMP choreographies have been proposed in chapter 3 as an B2Bi choreography style and an analysis framework that targets multi-party choreographies. The contents of this section, published in [189, 191], motivate the need for SeqMP, give its formal definition and define algorithms for solving common multi-party choreography issues.

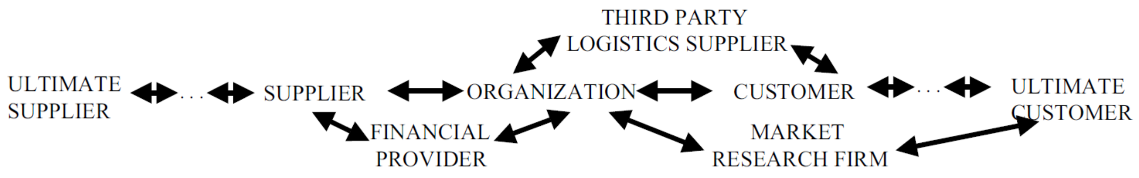


Figure 4.12.: Ultimate Supply Chain (taken from [112])

In the B2Bi domain, many approaches (for example [79, 186, 190, 232]) focus on strictly binary choreographies, i.e., on interactions between exactly two integration partners for very valid reasons (cf. sections 1.3, 4.4 and 4.6.1). While binary choreographies cover the majority of current B2Bi scenarios, multi-party scenarios actually are an implication of the concepts of supply chains/supply networks. Supply Chain Management (SCM) is the management of different types of business processes across the *complete* supply chain, *not only point-2-point* [90]. This is also reflected in the concept of the so-called *'Ultimate Supply Chain'* as given in figure 4.12 from [112] where supply chain partners interact with two or more other partners. Huemer and Hofreiter consistently argue [59] that interactions with more than one business partner at least have to be defined locally. Moreover, there are some real world examples that are not binary. For example, RosettaNet defines the so-called "Order-To-Cash With Logistics Service Provider Scenario"[4] depicted in figure 4.13. In this scenario, a Customer, a Supplier and a Logistics Service Provider (LSP) role (represented by BPMN 1.1 pools) are using RosettaNet PIPs (visualized as small cuboids labeled 3A4, 3A8 and so on) for exchanging business documents. Moreover, the local actions of each role for processing the business documents exchanged via PIPs are given. However, figure 4.13 only describes the intended flow of interactions and leaves out what happens if communication errors occur or if, for example, the Supplier and LSP role are not able to agree upon the provision of transportation services. Note that such technical/business errors only affect two of the three roles immediately (send and receive actions are defined for one role only). This raises the question whether or not erroneous behavior may have an effect on the remaining roles and how to detect problematic execution paths.
In order to provide a widely applicable solution to this problem, this section defines how multi-party choreographies can be composed from existing binary choreographies (section 4.6.1). Furthermore, the negative effect of technical/business errors between

[4] `http://www.rosettanet.org/Support/ImplementingRosettaNetStandards/eBusinessProcessScenarioLibrary/OrdertoCash/tabid/3320/Default.aspx`, last access: 12/20/2011

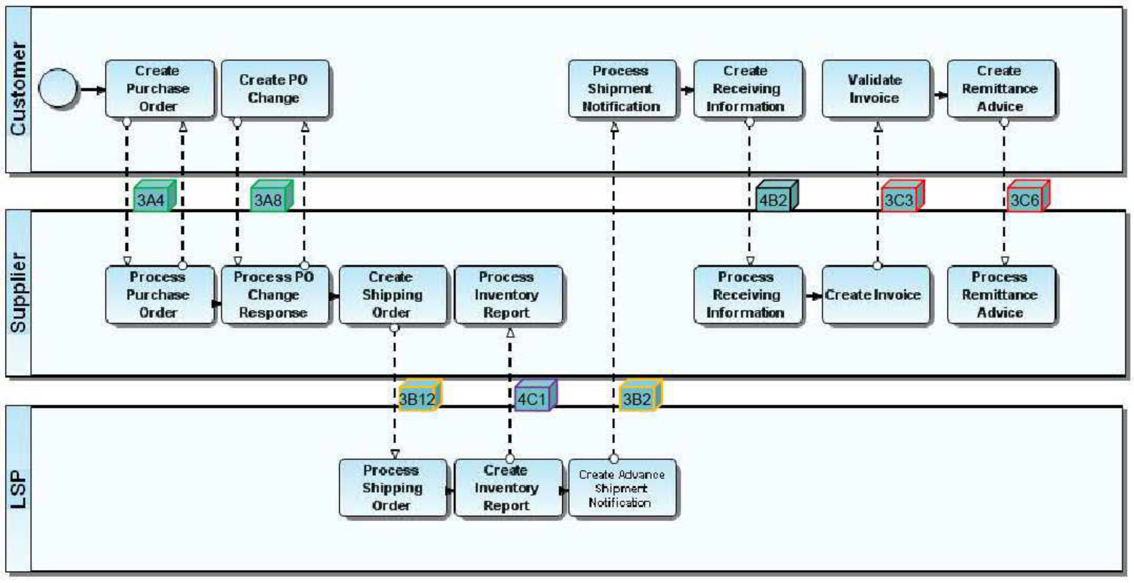


Figure 4.13.: RosettaNet Order to Cash with Logistic Service Provider Scenario (taken from RosettaNet[4])

two partners[3] on the remaining partners is captured as the so-called *partial termination problem* (section 4.6.2) and a configurable algorithm for identifying problematic execution paths is sketched (section 4.6.3). A second algorithm for deriving role projections from multi-party choreographies is given for fostering straightforward systems development. In addition, an algorithm for merging multiple projections (if existent) of the same role is given as well as rules for simplifying projections.

4.6.1. Definition

The class of SeqMP choreographies is tailored to the needs of B2Bi. By analyzing 100 scenarios of the publicly available *RosettaNet implementation guides* (for implementing B2Bi processes), the majority of interactions was discovered to be binary (84 scenarios), i.e., performed between exactly two integration partners. This is in line with academic research (cf. chapter 3). The remaining multi-party interactions of the analysis can be split up into binary interactions. Two factors can be identified that foster decomposability into binary interactions. Firstly, the atomic building blocks of many B2Bi processes are binary transaction-like concepts for the exchange of request business documents and optional response business documents. In the case of RosettaNet, PIPs are these atomic building blocks and despite the simple structure of PIPs the economic value exchanged using PIPs is worth billions of dollars (RosettaNet Standards Assessment 2008[4]). Remember that similar 'atomic building blocks' can also be

[3]The terms 'partner' and 'role' will be used interchangeably

[4]`http://www.rosettanet.org.my/Download/2009ImplementationStatistics05.26.09.pdf`, last access: 12/20/2011

found in ebBP, UMM [208] or BCL [248]. Secondly, the control flow defined typically is fairly simple, i.e., does not apply concepts like parallel structures or hierarchical decomposition. This, in turn, is in line with a multi-case study of Reijers et al. [166] who report the results of an investigation of 16 business processes from six Dutch organizations: *“Business processes turned out to be completely sequential structures. Their routing complexity was only determined by choice constructs and iterations.”* This finding is also backed by the B2Bi models created for the eBIZ-TCF project (`http://www.moda-ml.net/moda-ml/repository/ebbp/v2008-1/en/`) that do not use concurrent behavior (cf. section 4.4).
Now, as control flow of B2Bi interactions tends to be simple and the atomic building blocks are binary, multi-party choreographies can be viewed as sequences of binary choreographies, that means, as binary BCAs.
Figure 4.14 shows how binary BCAs can be used to model the use case of figure

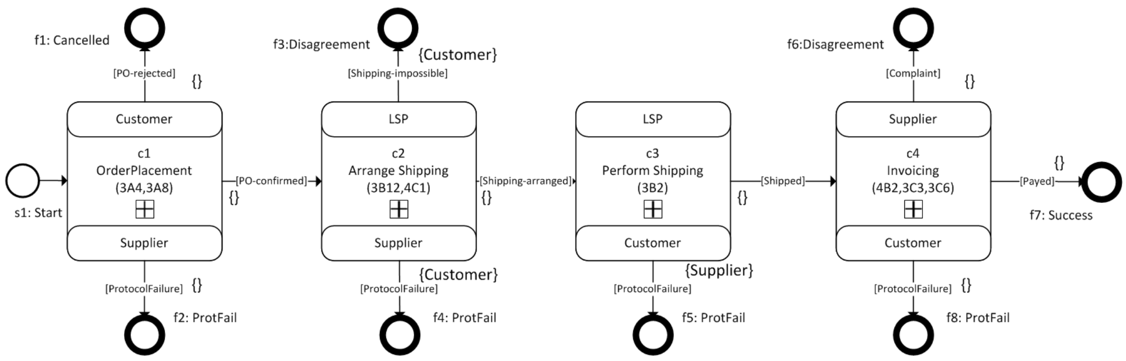


Figure 4.14.: SeqMP Model of the RosettaNet Use Case

4.13 in BPMN 2.0 [150] choreography notation (cf. chapter 6). Each binary BCA is composed of 1 to 3 PIPs as given in the original RosettaNet model (cf. figure 4.13). The binary choreographies (BCAs) are modeled as so-called BPMN *Collapsed Call Choreographies* and visualized as rounded rectangles with a ‘+’ at the bottom. The two bands at the top and at the bottom contain the integration partner roles participating in the call choreographies. The text in the middle contains an id (c1...c4), a name and the PIP types contained in the call choreographies (3A4, 3A8 and so on). Using the PIP types, it is easy to identify which call choreography corresponds to which part of the original RosettaNet choreography definition of figure 4.13. For defining the detailed structure of each binary call choreography, ebBP-ST and ebBP-Reg are ready for use.

The advantage of using binary BCAs as building blocks for multi-party choreographies is that integration partners can be assumed to have agreed upon the result of the binary BCA. Moreover, both partners start and terminate the BCA more or less in lock-step. Consequently, the result of the binary BCAs can be used for routing the control flow of the multi-party choreography. In figure 4.14, this is indicated by guards (expressions placed in brackets) that are attached to the transitions. The guard `[PO-confirmed]` after BCA `c1` captures confirmation of the purchase order exchanged whereas `[PO-rejected]` captures rejection. The corresponding transitions of these guards link to BCA `c2` or end state `f1` respectively. The annotations in curly braces are explained later. This concept of defining multi-party choreographies as sequentially performed binary BCAs with branching structures for defining control flow is reflected in the following definition.

Definition 4.6.1 (SeqMP Choreography)
A SeqMP choreography is a directed graph SeqMP (s_0, F,SBCA,T,R,RA) with the following elements:

- s_0 *the (unique) start state.*
- *F a non-empty set of final states.*
- *SBCA a non-empty set of binary BCAs.*
- *T the union of the following transition sets*
  - $T^{start} = \{(s_0, true, bca)\}$, *bca* $\in$ *SBCA*
  - $T^{end} \subseteq SBCA \times G \times F$
  - $T^{flow} \subseteq SBCA \times G \times SBCA$

  where G is a set of boolean guards consisting of the constants {true, else} and any disjunction of terms that consist of the names of the possible results of the BCA just performed. A term is evaluated upon termination of a BCA and becomes true when the BCA produces the corresponding result. 'else' becomes true if all guards of all other transitions with the same source become false.
- *R the set of roles of the SeqMP process.*
- *RA: SBCA* $\rightarrow R^2$*, a role assignment function that assigns exactly two roles to each BCA.* □

Note that, due to the fact that BCAs are executed sequentially, a SeqMP choreography basically is a state machine so that standard state machine semantics can be used to interpret it.

For characterizing the validity of SeqMP choreographies, the following auxiliary functions are defined.

Definition 4.6.2 (SeqMP Auxiliary Functions)

- *.-notation/#-notation is used for accessing the components of a tuple by name/index.*
- namesB *is the function that computes the names of the results of a BCA.*
- namesG *is the function that computes the names contained in a guard.*

- *A path* path*(a,b) between two nodes* $a,b \in \{s_0\} \cup F \cup SBCA$ *is a sequence of nodes* $a,n_{1..x},b$ *such that for all i=1...x-1,* $(n_i,g_i,n_{i+1}) \cup (a,g_a,n_1) \cup (n_x,g_x,b) \subseteq T$. *The length of a path(a,b)* length*(path(a,b)) is the number of nodes in the sequence. Let Path(a,b) be the set of all paths between a and b.* □

Based on this definition, it is possible to characterize the validity of SeqMP processes using the following three conditions.

Definition 4.6.3 (Valid SeqMP Choreography)
A SeqMP choreography smp is valid iff the following three conditions hold:

1. ***Subsequent role participation**:*
   $\forall$ $(s_1, g, s_2) \in smp.T^{flow}$: $RA(s_1) \cap RA(s_2) \neq \emptyset$, *i.e., for two subsequent BCAs at least one of the assigned roles must be the same (hence enabling synchronization between terminating one BCA and starting the next).*

2. ***Guard validity**:*
   Let $SUCC_{bca} \subseteq smp.T$ *be the set of outgoing transitions from some* $bca \in smp.BCA$ *with:* $\forall\ t\ (t\#1, t\#2, t\#3) \in SUCC_{bca}$: $t\#1 = bca$. *Then, the guards of bca are valid iff:*
   $(|SUCC_{bca}| = 1 \wedge \text{for } \{t\} = SUCC_{bca}: t\#2 = true) \vee$
   $(\forall\ t \in SUCC_{bca}: t\#2 \neq true \wedge$
   $(\ (\bigcup_{t \in SUCC_{bca}} namesG(t\#2) = namesB(bca)) \vee\ (\bigcup_{t \in SUCC_{bca}} namesG(t\#2) \subset namesB(bca) \wedge \exists\ t1 \in SUCC_{bca}: t1\#2 = else \wedge \nexists\ t2 \in SUCC_{bca}, t1 \neq t2: t2\#2 = else)\ ) \wedge$
   $\forall\ t3, t4 \in SUCC_{bca}, t3 \neq t4: namesG(t3\#2) \cap namesG(t4\#2) = \emptyset\ )$

3. ***Connectedness**:*
   $(\forall\ f \in smp.F: Path(smp.s_0, f) \neq \emptyset) \wedge \forall\ bca \in smp.SBCA: Path(smp.s_0, bca) \neq \emptyset \wedge \exists\ f \in smp.F: Path(bca, f) \neq \emptyset$. □

Actually, it would not be hard to extend this definition to using multi-party BCAs as building blocks (and even the algorithms in section 4.6.3 would work) as long as an agreed-upon result among all participants of the BCAs would be guaranteed. This, however, does not seem to hold true for many real-world scenarios.

4.6.2. Problems in Multi-Party Choreographies

Remodeling the RosettaNet Order-To-Cash use case as depicted in figure 4.14 immediately reveals two important problems, *partial termination* and creation of *role projections.*
Partial termination becomes obvious when looking at the transitions that lead into final states `f3`, `f4` and `f5` of figure 4.14. As the source states of these transitions are binary BCAs, only those roles participating in the respective BCA will be aware of the termination of the overall SeqMP choreography. However, the *Customer* role (in case of BCA `c2` *Arrange Shipping*) or the *Supplier* role (in case of BCA `c3` *Perform*

Shipping) may still wait for some interaction to happen. This may not be a problem in case the individual BCAs of a SeqMP choreography are independent of each other, but the business semantics of RosettaNet's Order-To-Cash scenario contradicts independence of, for example, `c1` and `c4`. One possibility to attack this problem is adding additional BCAs implementing exception handling routines. However, modeling such multi-party choreographies may be hard because such exception handling BCAs may fail as well and suitable business documents for communicating exception handling semantics are not always available in business document libraries. Furthermore, a business level problem like disagreeing on the conditions of shipping between *Supplier* and *LSP* may require business escalation routines between *Supplier* and *Customer* that are not intended to be implemented using business-document based choreographies. In essence, not defining explicit handling routines for arbitrary exceptional circumstances is a natural thing in process specification and hence partial termination is an integral problem of multi-party processes. Note that the different representations of the same use case in figures 4.13 and 4.14 do not have an influence on the actual existence of the partial termination problem. It is just not as obvious in the original RosettaNet representation of figure 4.13 because only the *intended* flow of interactions is modeled.
Creation of *role projections* is an obvious problem when considering that some roles may not participate in every BCA of a SeqMP choreography. Hence, the possible sequences of BCA executions with participation of a particular role r must be derived from the overall choreography and the BCAs without participation of r must be suitably abstracted.
Both problems are not hard to solve if the use case is as simple as in figure 4.14. However, other SeqMP choreographies such as the artificial use case depicted in figure 4.15 may be more challenging. The use case depicts a multi-party order-to-cash choreography between a *Customer*, a *Seller*, a *Logistics Service Provider* (LSP), a *Financial Service Provider* (FSP) and an *Escort Service Provider* (ESP). For compactness, the BPMN choreography notation has been conflated in an ad-hoc manner by only showing the participating roles and an id for each call choreography. The business semantics of the use case may be derived to some extent from the guards on the transitions, but, for the purpose of discussing SeqMP, the control flow of the use case is decisive. One striking observation is that *partial termination* is not a problem associated to final states only, but actually to transitions that partition a SeqMP choreography in parts with or without possible participation of a particular role. For example, by firing the transition between *c6* and *c8* as depicted in figure 4.15, there is no possibility that the *FSP* role will become active in the particular SeqMP instance anymore whereas participation still would be possible in *c6*.

4.6.3. SeqMP Algorithms

This section presents algorithms for dealing with the *partial termination* and the *role projection* problem identified in the last section. Note that the algorithms are defined for SeqMP choreographies which are based on binary BCAs as building

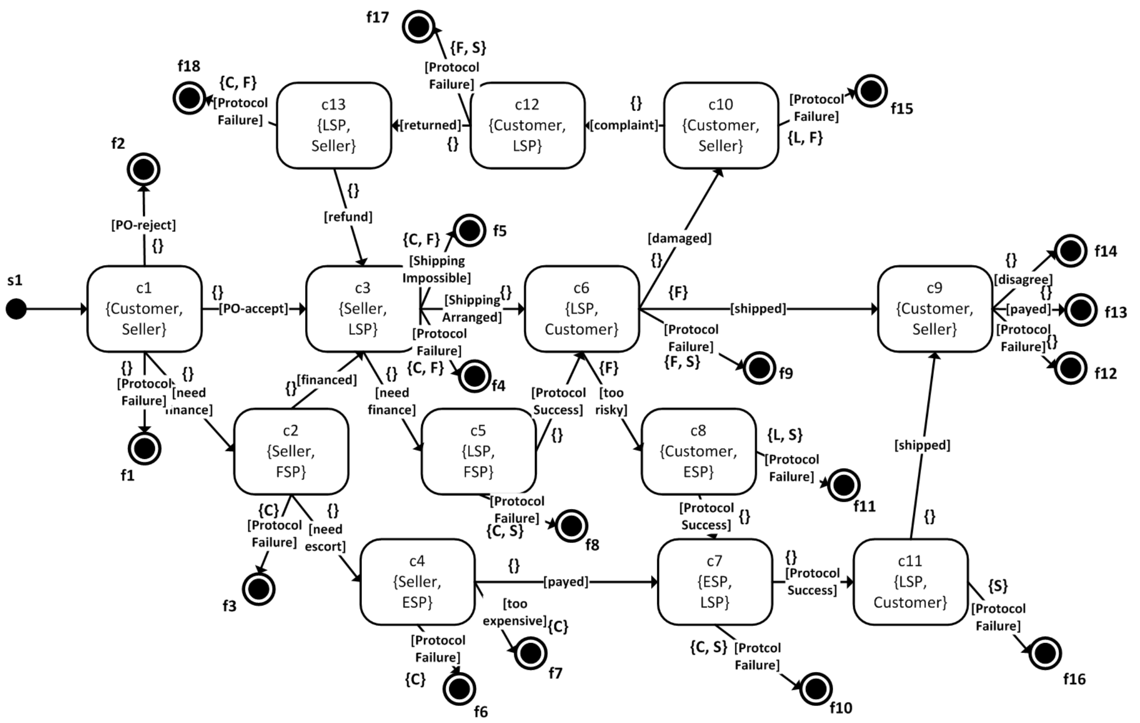


Figure 4.15.: SeqMP Model of a Complex Use Case (conflated visualization)

blocks. However, the algorithms themselves also would work for multi-party BCAs as building blocks. For validation, the algorithms for calculating escalation sets as well as role projections have been implemented for an abstract model of SeqMP in Java. The prototype implementation together with the test graphs for figure 4.14 and 4.15 are available[5].

4.6.3.1. Computing Escalation Sets

As really safe multi-party choreographies that specify exception handling logic in full (cf. above) are hard to design, the computation of so-called *escalation sets* is proposed for tackling the *partial termination* problem. Intuitively, escalation sets are sets of participation expectations that cannot be fulfilled any more upon firing a particular transition. Consider the transition (c2-f4) of figure 4.14 again. The *Customer* has participated in BCA c1 and then waits for BCA c4 to begin. So, a participation expectation has been created for the Customer. When firing (c2-f4), this expectation cannot be satisfied any more, but it could in c2. In addition, the Customer does not participate in c2 and so she is not informed about that. Hence, firing (c2-f4) creates a partial termination problem for the Customer and the string 'Customer' is added to the escalation set of (c2-f4) (represented as curly braces added to the transition in figure 4.14) to specify that fact. For analogous reasons, 'Customer'

[5] `http://www.uni-bamberg.de/pi/confSeqMP-Algorithms`, last access: 12/20/2011

is added to (c2-f3) and 'Seller' is added to (c3-f5). To capture this intuition more precisely, the concepts of 'Expectation' and 'Escalation Assignment' are defined.

Definition 4.6.4 (Expectation)
An expectation is a 2-tuple E(r,en) where $r \in smp.R$ for some SeqMP choreography smp and en is a name for the expectation. □

Definition 4.6.5 (Escalation Assignment)
An escalation assignment is a function ESA: $smp.T \rightarrow 2^E$ that, for a SeqMP choreography smp and a set of expectations E, assigns to each transition $t \in smp.T$ a set of expectations $se \subseteq E$. □

Then, informally, escalation sets can be characterized as follows:

> *If an expectation (r,en) may have been created on some path to BCA bca and if the expectation still may be satisfied by some reachable BCAs and transitions and if the expectation may not be satisfied any more by taking a particular outgoing transition of bca and if $r \notin RA(bca)$ (because r would have full information otherwise) then the expectation goes into the escalation set of that transition.*

Note that this informal definition does not refer to the concrete path that is taken at run-time for determining expectations. Instead, all paths that *may* have been taken for reaching bca are considered. That means that the participants of a SeqMP choreography can find out at design time which transitions potentially suffer from the *partial termination* problem at runtime. Based on this information, they may agree on appropriate actions for dealing with this issue, e.g., informing their integration partners via mail or phone. In so far, escalation sets are a means for analyzing the partial termination problem in multi-party choreographies, but not a fully automatic solution to the problem. Furthermore, when firing a problematic transition at runtime the defined escalation set has to be compared to the actual expectations created during that particular process instance (which could easily be implemented by some logging feature).

So far, the details of when expectations are created and when these are satisfiable have not been discussed. Three distinct strategies (or modes), namely 'ALWAYS', 'SELECTED', and 'RESOLVABLE', can be identified for accomplishing this task. The escalation sets then can be computed according to the 'Escalation Set Computation' algorithm as specified in algorithm objects 4, 5 and 6. Below, the different strategies are discussed in more detail:

Strategy ALWAYS:

This strategy basically assumes that an expectation is created whenever a role r participates in a BCA and that it is satisfiable as long as r may still participate in some future BCA. The escalation criterion is (relative to some SeqMP smp) for

Algorithm 4: Compute Escalation Set: Part1

```
input      :
smp, a valid SeqMP choreography;
mode, the operating mode;
rsa, a role selection assignment //SELECTED mode;
epa, an expectation assignment //RESOLVABLE mode;
rea, a resolution assignment //RESOLVABLE mode;
output     :
esa, an escalation assignment;
variables  :
// Maps for capturing reachable states/transitions;
Map<State,Set<State>> mapStFwd, mapStBwd;
Map<State,Set<Trans>> mapTrFwd, mapTrBwd;
// Result map for mapping transitions to expectations;
Map<Trans,Set<Expectation>> mapEsc;
procedure : compExpect(State s) :
if mode = ALWAYS then
    Set<State> bwds = mapStBwd.get(s);
    return $\bigcup_{st \in bwds}$RA(st) × {'ALWAYS'};
else if mode = SELECTED then
    Set<Trans> bwdtr = mapTrBwd.get(s);
    return $\bigcup_{t \in bwdtr}$rsa(t) × {'SELECTED'} × t.id();
else if mode = RESOLVABLE then
    Set<Trans> bwdtr = mapTrBwd.get(s);
    return $\bigcup_{t \in bwdtr}$epa(t);
end
end procedure // continue...
```

some t ∈ smp.T:
ESA(t(t#1,t#2,t#3)) = {(r,'ALWAYS') | (∃ path(s_1, t#1), s_1 ∈ smp.SBCA, length(path) > 1: r ∈ RA(s_1)) ∧ (∃ path(t#1, s_2), s_2 ∈ smp.SBCA, length(path) > 1: r ∈ RA(s_2)) ∧ (∀ path(t#3, s_3), s_3 ∈ smp.SBCA, length(path) > 1: r ∉ RA(s_3)) }
The advantage of this strategy is that users can apply this strategy to a valid SeqMP choreography as is, i.e., no additional configuration is needed. The disadvantage is that the strategy ignores that the execution of some BCAs does not create expectations whereas others do.
The result of applying this strategy is reflected in the use cases of figures 4.14 and 4.15 where the escalation sets are attached to each transition by enumerating the corresponding roles in curly braces. For figure 4.15, only the initial letters of each role are given and the string 'ALWAYS' is left out for presentation purposes. For example, the escalation set {F} of the transition between `c6` and `c9` of figure 4.15 says for the Financial Service Provider (FSP) role that it may have participated in an instance

Algorithm 5: Compute Escalation Set: Part2

```
// continue...
procedure: compResolve(Set<Expectation> resolveSet, State s, Trans t) :
Set<Expectation> theRes = new Set();
State search = s;
if $t \neq null$ then search = t #3;
foreach Expectation e in resolveSet do
    if mode $\in$ {ALWAYS,SELECTED} OR (mode = RESOLVABLE AND rea(e) = $\emptyset$) then
        // check resolution via role participation
        Set<State> fwds = mapStFwd.get(search);
        if $t \neq null$ then fwds.add(t #3);
        Set<Role> roles = $\bigcup_{st \in fwds}$RA(st);
        if e #1 $\in$ roles then theRes.add(e);
    else if mode = RESOLVABLE AND rea(e) $\neq \emptyset$ then
        // check resolution via reachable transitions
        Set<Trans> fwdtr = mapTrFwd.get(search);
        if $t \neq null$ then fwdtr.add(t);
        if $\exists$ res $\in$ rea(e). res $\subseteq$ fwdtr then
            theRes.add(e)
        end

end
return theRes;
end procedure // continue...
```

of the depicted SeqMP graph (if `c2` or `c5` were on the path to `c6`), that the FSP role still could be triggered (via the path (`c6`,`c10`,`c12`,`c13`,`c3`,`c5`)) and that it cannot be triggered any more when the transition has been fired (because from `c9` no BCA with participation of FSP can be reached). This may or may not be an issue depending on whether or not the FSP considers each BCA `c5` instance as a completely independent business case. The LSP role, in turn, is not included in the escalation set between `c6` and `c9` because it knows from the `[shipped]` outcome and the global SeqMP model that there is no way of being involved in the current SeqMP instance any more.

For the next strategy the concept of 'Role Selection Assignment' is introduced:

Definition 4.6.6 (Role Selection Assignment)
A role selection assignment is a function RSA: $smp.T \rightarrow 2^{smp.R}$ that, for a SeqMP choreography smp, assigns to each transition $t \in smp.T$ the set of roles $sr \subseteq \{r |\ r \in RA(t\#1)\}$ for which the expectation to participate once more during the choreography is created upon firing t. □

Algorithm 6: Compute Escalation Set: Part3

```
algorithm :
// continue...
foreach State s in in a traversal of smp do
    // Compute states/transitions reachable in forward/backward direction; s is not part of
       the reachability set
    mapStFwd.put(s,compStateFwd(s));
    mapStBwd.put(s,compStateBwd(s));
    mapTrFwd.put(s,compTransFwd(s));
    mapTrBwd.put(s,compTransBwd(s));
end
// Actual escalation assignment calculation
foreach State s in in a traversal of smp do
    Set<Expectation> eSet = compExpect(s);
    // Remove entries of roles participating in s
    foreach (r,e) ∈ eSet do
        if r ∈ RA(s) then eSet = eSet \ {(r,e)};
    end
    // Determine resolvable expectations from s onwards
    Set<Expectation> allRes = compResolve(eSet,s,null);
    // Compare allRes to resolvable expectations of outgoing transitions
    foreach Trans t ∈ $SUCC_S$ do
        Set<Expectation> trRes = compResolve(allRes,null,t);
        if allRes = trRes then
            mapEsc.put(t,new Set());
        else
            Set<Expectation> noRes = copy(allRes);
            noRes = noRes \ trRes;
            mapEsc.put(t,noRes);
        end
    end
end
return mapEsc;
```

Strategy SELECTED:

This strategy is similar to the 'ALWAYS' strategy but not every outgoing transition of a BCA creates an expectation but only those transitions that the user configures by specifying a role selection assignment. In addition, the user has the option to specify for which of the participating roles of a BCA a particular outgoing transition creates an expectation. Expectations are satisfiable as long as r may still participate in some future BCA. The escalation criterion is (relative to some SeqMP smp and role selection assignment RSA) for some t,t′ ∈ smp.T:
ESA(t(t#1,t#2,t#3)) = {(r,'SELECTED'+t.id()) | (∃ path(s_1, t#1), s_1 ∈ smp.SBCA:

r ∈ RSA(t′) with t′#3 = s_1) ∧ (∃ path(t#1, s_2), s_2 ∈ smp.SBCA, length(path) > 1: r ∈ RA(s_2)) ∧ (∀ path(t#3, s_3), s_3 ∈ smp.SBCA, length(path) > 1: r ∉ RA(s_3)) }
The advantage of this strategy is that it imposes a moderate configuration burden on the user while offering the possibility to exclude transitions from expectation creation. The disadvantage of this strategy is that the user has no control about the events that lead to the satisfaction of an expectation.

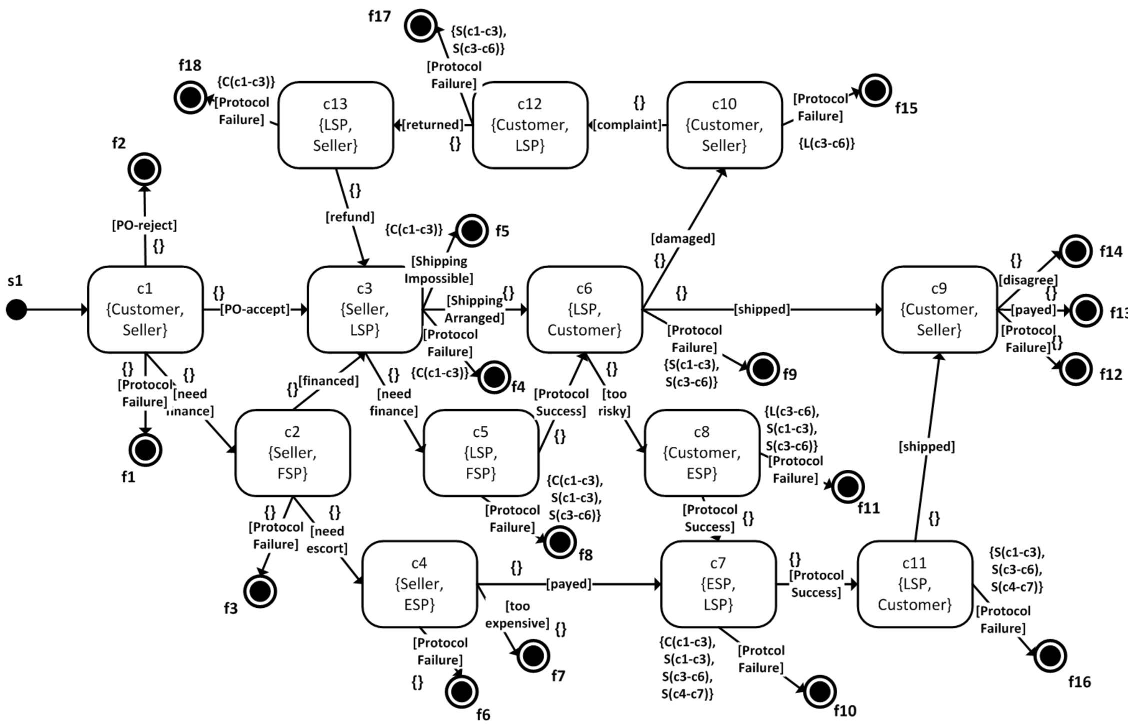


Figure 4.16.: SELECTED Strategy Applied to the Use Case of Figure 4.15

Figure 4.16 shows the result of applying the SELECTED strategy to the complex use case of figure 4.15 with the following role selection assignment RSA:
RSA = {(c1-c3,{customer, seller}), (c3-c6,{esp, seller}), (c4-c7,{lsp, seller})}
There are three major differences to observe when comparing this result to the result of the ALWAYS strategy as shown in figure 4.15: Firstly, the Customer role is not included in the escalation sets of transitions (c2-f3), (c4-f6) and (c4-f7) because only transition (c1-c3) is configured to create an expectation for the Customer and this transition is not reachable in backward direction of those transitions. Secondly, as there are no expectations configured for the Financial Service Provider this role is not included in any escalation set. Thirdly, multiple expectations are created for the Seller role. The transitions have been added to the role names in figure 4.16 to distinguish between the various expectations, e.g., 'S(c1-c3)', 'S(c3-c6)' or 'S(c4-c7)'. The escalation sets then reflect which of those expectations potentially are violated. So, there are three violated Seller expectations for (c11-f16) whereas there are only two violated Seller expectations for (c12-f17). Note that these results

highly depend on the role selection assignment provided by the user and therefore the business meaning of escalation sets is heavily influenced by the user. For example, not configuring an expectation for the FSP role may mean that the BCAs with FSP participation are completely independent of other BCAs or it may mean that the user just did not want to focus on the FSP role.

For the last strategy, the definitions of ‘Expectation Assignment’ and ‘Resolution Assignment’ are needed:

Definition 4.6.7 (Expectation Assignment)
An expectation assignment is a function EPA: smp.T → 2^E that, for a SeqMP choreography smp and a set of expectation names EN, assigns to each transition t ∈ smp.T the set of expectations se ⊆ E={(r,en)| r ∈ RA(t#1), en ∈ EN} that is created upon firing t. □

Definition 4.6.8 (Resolution Assignment)
A resolution assignment is a function REA: E → $2^{2^{smp.T}}$ that, for a SeqMP choreography smp and a set of expectations E, assigns to each expectation e ∈ E the set of sets of transitions setRes ⊆ $2^{smp.T}$ for which each element resolves e. □

Strategy RESOLVABLE:

The last strategy offers the possibility to create multiple expectations per role upon firing a transition and to define sets of sets of transitions that need to be fired to satisfy an expectation. The escalation criterion is (relative to some SeqMP smp, expectation assignment EPA and resolution assignment REA) for some t,t′ ∈ smp.T:
ESA(t(t#1,t#2,t#3)) = {ep ∈ EPA(t′) | (∃ path(s_1, t#1), s_1 ∈ smp.SBCA: t′#3 = s_1) ∧ (∃ a sequence of transitions t_1,...,t_n ∈ rp.T. (t_1#1 = t#1 ∧ ∃ reSet ∈ REA(ep). reSet ⊆ t_1,...,t_n)) ∧ (∄ a sequence of transitions t_1,...,t_n ∈ rp.T. (t_1#1 = t#3 ∧ ∃ reSet ∈ REA(ep). reSet ⊆ t_1,...,t_n)) }
While this strategy places considerable burden upon the user to define comparatively complex expectation and resolution assignments, it allows for fine-grained configuration possibilities to define when expectations are created and satisfied. In addition, users may choose to avoid the specification of complete expectation and resolution assignments by focusing in on only selected expectations and resolutions.

Figure 4.17 shows the result of applying the RESOLVABLE strategy to the complex use case of figure 4.15 with the following expectation assignment EPA and resolution assignment REA:
EPA =
{(c1-c3,{(Customer,ReceiveProduct), (Customer,PayProduct)}),
(c1-c2,{(Customer,EscortedProduct)}),
(c6-c10,{(Customer,ResolveDamage)}),
(c10-c12,{(Customer,SendBack)}),
(c8-c7,{(Customer,ReceiveEscort)})}

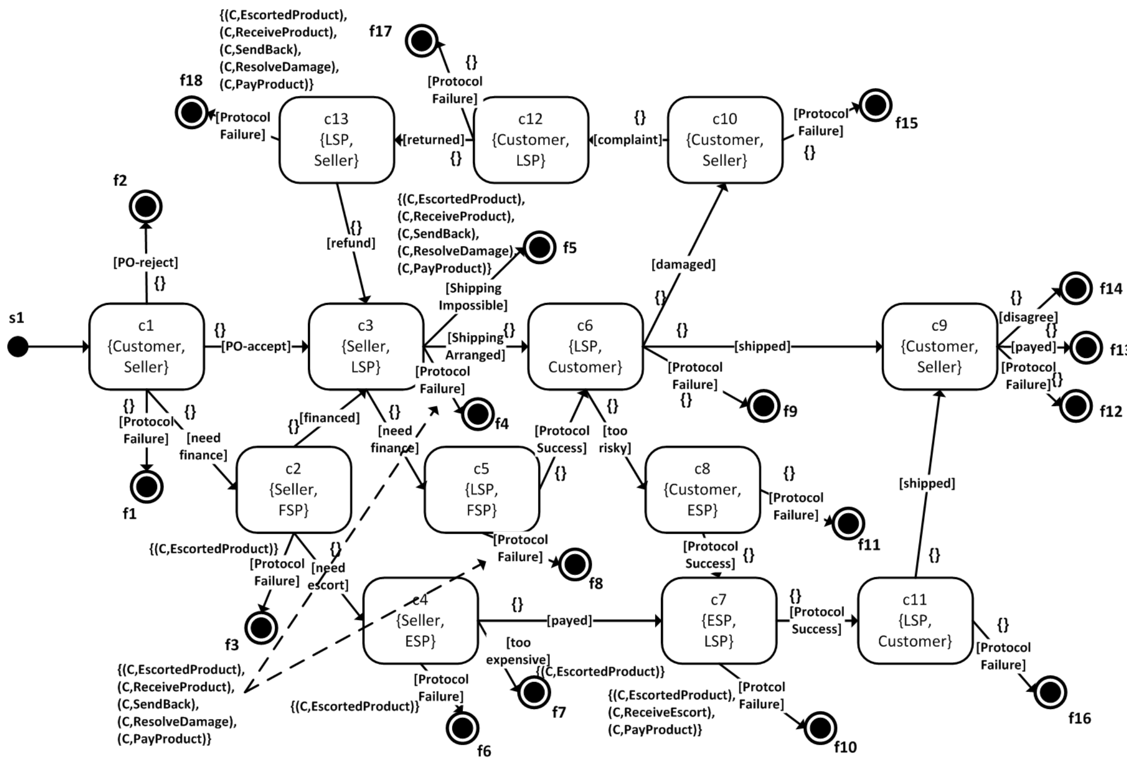


Figure 4.17.: RESOLVABLE Strategy Applied to the Use Case of Figure 4.15

REA =
{((Customer,ReceiveProduct),{{(c6-c8)},{(c6-c9)}}), ((Customer,PayProduct),{{(c9-f13)}}),
((Customer,EscortedProduct),{{(c11-c9),(c9-f13)}}),
((Customer,ResolveDamage),{{(c6-c8)},{(c6-c9)}}),
((Customer,SendBack),{{(c12-c13)}}),
((Customer,ReceiveEscort),{{(c11-c9)}})}

When comparing the RESOLVABLE strategy to the first two strategies observe the following extensions: Firstly, there may be multiple expectations per transition for the same role. For example, upon firing (c1-c3) the expectations named ‘ReceiveProduct’ and ‘PayProduct’ are created for the Customer role. Secondly, the resolution of expectations is configurable and does not depend on role participation during BCAs. For example, transition (c9-f13) is configured to resolve the PayProduct expectation whereas either (c6-c8) or (c6-c9) can resolve ReceiveProduct. Therefore, (Customer, PayProduct) is part of the escalation set of (c7-f10) whereas (Customer, ReceiveProduct) is not. The reason is that neither (c6-c9) or (c6-c8) are reachable in forward direction from BCA c7. So firing (c7-f10) does not cut off any resolution of ReceiveProduct and therefore ReceiveProduct does not go into the escalation set of (c7-f10). Note that the result of the RESOLVABLE strategy highly depends on the expectation and resolution assignment provided by the user and thus allows a fine-grained analysis of SeqMP choreographies.

4.6.3.2. Computing Role Projections

Role projections of multi-party choreographies are useful for focusing on the relevant behavior of a particular role *r*. The approach for computing projections of a SeqMP *smp* for r is based on abstracting interactions without participation of *r* using so-called event-based choices (EBCs). So, if a BCA without participation follows a BCA with participation of r then the second BCA is replaced by an EBC. The EBC then is the new target of the transition between the two BCAs. Conversely, if a BCA with participation of r follows a BCA without participation then the transition between the two BCAs is removed and a new transition between the EBC for abstracting the former BCA and the BCA with participation of r is created. In figures 4.18, 4.19, 4.20 and 4.21, black vertical bars are used to represent EBCs. Consider figure 4.18 which shows the projection of the use case of figure 4.15 for the Customer role. As the Customer does no participate in BCA c3, it is abstracted by EBC ebc1 and transition (c1-c3) is linked to ebc1. c5 is then conflated with ebc1 because the Customer does not participate in c5 either. ebc1 then is linked to BCA c6 as this is the next BCA with participation of the Customer. The guard of this transition is "true" because it is transparent for the Customer what happens in the meantime. Note that EBCs may also be created if the predecessor is not a BCA with participation of the focal role. For example, BCAs c2 and c4 are represented by ebc4. When processing transition (c4-c7) an extra EBC (ebc3) is created because c7 is reachable via c8 as well. Merging ebc4 and ebc3 would put the behavioral integrity of the projection at risk because then all outgoing transitions of ebc4 would be reachable via c8 as well. Note that in [189], EBCs were merged in such a case and an additional EBC simply was introduced for the path via c8. The behavior of the algorithm was changed because the new algorithm creates more compact projections and provides that the projection of a state is independent of the path through which it is discovered.

Definitions 4.6.9 and 4.6.10 formally capture role projections and some auxiliary functions.

Definition 4.6.9 (Role Projection)
A role projection rp_r for role r of a SeqMP choreography smp is a directed graph Proj (s_0, S, T) with the following elements:

- $s_0 = smp.s_0$ *the (unique) start state.*
- $S = s_0 \cup EBC \cup F \cup SBCA$ *the states of the projection with EBC a set of event-based choice states,* $F \subseteq smp.F$, $SBCA \subseteq smp.SBCA$ *with* $\forall\ bca \in SBCA.\ r \in smp.RA(bca)$.
- $T \subseteq S \times smp.G \times S$ *the set of transitions where for each* $t \in T.\ t \in EBC \times \{true\} \times EBC \vee t \in smp.T \vee (\exists t' \in smp.T.\ (t{=}(t'\#1,t'\#2,e) \vee t{=}(e,true,t'\#3)) \wedge e \in EBC)$. □

Definition 4.6.10 (Projection Auxiliary Functions) *The auxiliary functions defined for SeqMP choreographies in definition 4.6.2 are correspondingly defined for*

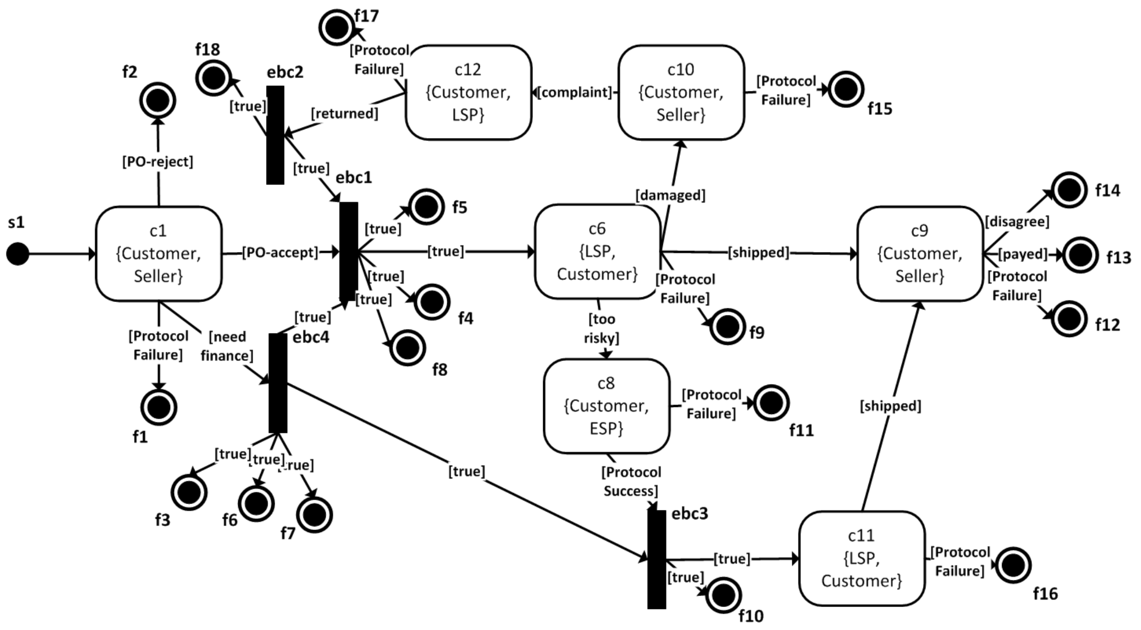


Figure 4.18.: Projection for the Customer Role (Use Case of Figure 4.15)

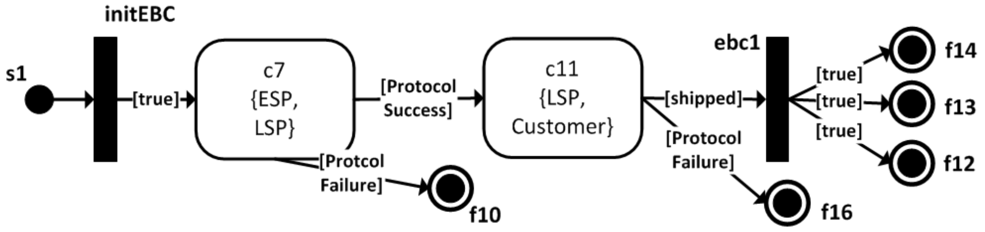


Figure 4.19.: 1 out of 2 Possible Projections for the LSP Role (Use Case of Figure 4.15)

role projections. In particular, a path$_r$*(a,b) is a path between two nodes a and b in projection rp and* $Path_r(a,b)$ *is the set of all paths between a and b in projection rp. If* $rp.s_0$ *links to an event-based choice then* rp.initEBC *can be used to refer to that event-based choice.* □

The role projection algorithm presented in algorithm objects 7, 8 and 9 gives the details of computing role projections. Note that there may be more than one projection per role. Multiple projections of a particular SeqMP *smp* for a particular role *r* are deliberately allowed for because different branches of a multi-party choreography potentially may comprise completely unrelated BCAs of r. Consider figure 4.15 and the LSP role. If an instance of the choreography began with BCAs c1, c2 and c4 then only the BCAs of figure 4.19 would be possible for the LSP role. Assume further there was no transition between c8 and c7. Then, depending on whether transition (c2-c4) or (c2-c3) is taken, completely disjoint sets of BCAs with LSP-participation are possible. In that situation, integration participants should not be forced to view unrelated BCAs as part of a single collaboration. Algorithm 7 can be used for identifying potentially disjoint projections for a particular role r. Once the first BCA with participation of r is found that could be the starting point of such a projection, the computation of a role-specific projection at a particular state is started (algorithm objects 8 and 9). For the case of the LSP role of the use case of figure 4.15, this approach leads to two individual projections that overlap. In this case, the individual projections still may be considered to be two different use cases or may be merged which depends on the system setting of the integration participant. Algorithm object 10 shows how two distinct projections of the same role can be merged such that the individual projections and the corresponding merge are behaviorally equivalent. The algorithm is defined for two projections, but it can be applied iteratively to cover the case of more than two projections.

Algorithm 7: Projection Computation

```
input     :
A valid SeqMP smp to be analyzed
output    :
A mapping of roles to their projections: smp.R → Set<State>
variables :
Set<State> computed;
//initially maps each role of smp to an empty set of projections;
Map<Role,Set<State>> projs;
Boolean processBegin;
algorithm:
1  processBegin = true;
2  foreach State s of each path without loop in a depth first traversal of smp do
3      Set<Role> known = $\bigcup_{r \in \text{states on the current path} \setminus s} \texttt{RA}(r)$;
4      Set<Role> curr = RA(s);
5      curr = curr \ known;
6      if curr ≠ ∅ ∧ s ∉ computed then
7          foreach Role r in curr do
               // Add role projection for r starting at s
8              projs.get(r).add(doProjection(r, s, processBegin));
9          end
10         computed.add(s);
11     end
12     if processBegin then processBegin = false;
13 end
14 return projs;
```

Algorithm 8: Role Projection - State s: Part 1

```
input     :
State s and Role r of a valid SeqMP smp;
boolean first, true if s is the first BCA in smp
output    :
The first state of the role projection $s_{out}$
variables :
Map<String, State> proj //state ids to projected states;
Map<State,State> visitMap //source states to projected states;

procedure: walk(State curr) :
1  State mSelf = visitMap.get(curr);
2  foreach Trans t =(t #1,t #2,t #3) in $SUCC_{curr}$ do
3      State mTarg = visitMap.get(t #3);
4      if mTarg $\neq$ null then
5          if mSelf is an EBCState AND mTarg $\notin$ $SUCC_{mSelf}$ then //define transition
           between #1 and #3 with guard #2
6              trans(mSelf, "true", mTarg);
7          else
8              trans(mSelf, t #2, mTarg);
9          end
10     else
11         if t#3 is a FinalState $\vee$ r $\in$ RA(t#3) then //copy state
12             State nextCopy = t#3.scopy();
13             visitMap.put(t#3, nextCopy);
14             proj.put(t#3.id(), nextCopy);
15             if mSelf is an EBCState then
16                 trans(mSelf, "true", nextCopy);
17             else
18                 trans(mSelf, t#2, nextCopy);
19             end
20         else //abstract by event-based choice
21             if mSelf is an EBCState then
                   // alternative path to t #3?
22                 if $\exists$ bca $\in$ smp.BCA. r $\in$ RA(bca) $\wedge$ Path(bca,t #3) $\neq \emptyset$ then
23                     EBCState ebc = new EBCState(); visitMap.put(t#3, ebc);
24                     proj.put(ebc.id(), ebc);
25                     trans(mSelf, "true", ebc);
26                 else
27                     visitMap.put(t#3, mSelf);
28                 end
29             else
30                 EBCState ebc = new EBCState(); visitMap.put(t#3, ebc);
31                 proj.put(ebc.id(), ebc);
32                 trans(mSelf, t#2, ebc);
33             end
34         end
35         walk(t#3);
36     end
37 end
38 end procedure // continue...
```

Algorithm 9: Role Projection - State s: Part 2

```
algorithm:
// ...continue
$s_{out}$ = new StartState("s1");
State currCopy = s.scopy() // copy s without transitions
proj.put("s1", $s_{out}$);
proj.put(s.id(), currCopy);
if first then
    trans (proj.get("s1"), "true", proj.get(s.id()));
else
    proj.put("ebc0", new EBCState("ebc0"));
    // define transition between #1 and #3 with guard #2
    trans ($s_{out}$, "true", proj.get("ebc0"));
    trans (proj.get("ebc0"), "true", proj.get(s.id()));
end
visitMap.put(s, currCopy);
walk(s);
return $s_{out}$;
```

Algorithm 10: Projection Merge

```
input     :
State $rp_1$, $rp_2$, two projections of the same role;
output    :
State $rp_m$, the merged role projection;
algorithm :
1  $rp_m = rp_1$.deepCopy()// copy of complete projection of $rp_1$
   // insert extra leading event-based choice to avoid new traces by loops back to $rp_1.initEBC$
2  State $mergeEBC$ = new() EBCState();
3  $rp_m.S$.add($mergeEBC$);
   // use prj($state$) to get the copy of state
4  $rp_m.T$.del((prj($rp_1.s_0$),"$true$",prj($rp_1.initEBC$)));
5  $rp_m.T$.add((prj($rp_1.s_0$),"$true$",$mergeEBC$));
6  $rp_m.T$.add(($mergeEBC$,"$true$",prj($rp_1.initEBC$)));
   // prepare adding $rp_2$
7  $rp_m.T$.add(($mergeEBC$,"$true$",$rp_2.initEBC$.copy()));
8  foreach $Trans\ t$ =($t$ #1,$t$ #2,$t$ #3) in $a$ breadth first traversal starting from
   $rp_2.initEBC$ do
9      if $t$ #3.id() $\notin$ $rp_m.S$.ids() then
10         $rp_m.S$.add($t$ #3.copy());
11     end
       // copy transitions
12     $rp_m.T$.add({(prj($t$ #1),$t$ #2,prj($t$ #3))});
       // Stop upon encountering an overlap node
13     if $t$ #3.id() $\in$ $rp_1.S$.ids() then
14         stopTraversalBranch();
15     end
16 end
17 return $rp_m$;
```

For reasoning about behavioral equivalence, definitions 4.6.11 and 4.6.12 introduce the concepts of 'Boundary Overlap Node' and 'Execution Traces'. Theorem 4.6.1 then formally captures behavioral equivalence and lemmata 4.6.2, 4.6.3 and 4.6.4 are used during its proof.

Definition 4.6.11 (Boundary Overlap Node)
A boundary overlap node of two role projections rp_1 and rp_2 is a BCA bca such that $bca \in rp_1.SBCA \wedge bca \in rp_2.SBCA \wedge$
$($
$(\nexists bca' \in rp_2.SBCA.\ \exists\ path(rp_1.s_0,bca'),\ path(bca',bca).\ bca \in path(rp_1.s_0,bca')) \vee$
$(\nexists bca' \in rp_1.SBCA.\ \exists\ path(rp_2.s_0,bca'),\ path(bca',bca).\ bca \in path(rp_2.s_0,bca'))$
$)$. □

A boundary overlap node also can be informally characterized as a node that is part of both projections under consideration that is not preceded on any path from the respective start states to itself by another node shared by the two projections.

Definition 4.6.12 (Execution Traces)
An execution trace trace*(a,b) between two states a and b of a role projection rp is a sequence of states $a,st_0,\ldots,st_n,b$ that is derivable from some path(a,b) by removing all event-based choices. Let Trace(a,b) be the set of all traces between a and b. For some state $s \in rp.S$ of projection rp, the set of* leading *traces is $Trace^{lead}(s) = Trace(rp.s_0,s)$ and the set of* trailing *traces is $Trace^{trail}(s) = \bigcup_{f \in rp.F} Trace(s,f)$.* □

Theorem 4.6.1 (Behavior Preservation of Merge) *Consider two non-identical projections rp_1 and rp_2 for the same role r of a valid SeqMP choreography smp and the merge rp_m of rp_1 and rp_2 as derived by algorithm 10. Then $Trace^{trail}(rp_m.s_0) = Trace^{trail}(rp_1.s_0) \cup Trace^{trail}(rp_2.s_0)$.* □

Proof 4.6.1
The proof is split up into two parts:
Part 1: *rp_m produces all traces contained in rp_1 and rp_2.*
By lemma 4.6.4 and the definition of execution traces, it is sufficient to consider the forward subgraphs of rp_1, rp_2 starting with $rp_1.initEBC$, $rp_2.initEBC$ respectively. The subgraph starting with $rp_1.initEBC$ is a subgraph of rp_m. So, rp_m produces all traces of rp_1. Furthermore, let B be the set of boundary nodes of rp_1 and rp_2. By algorithm 10, lines 9 to 12, all states and transitions not connected to a boundary node are copied to rp_m without modification. Otherwise, consider an arbitrary path $p = rp_2.initEBC,\ldots,bca',\ldots,b$ with $b \in B$. If $bca' \in rp_1.SBCA$ then bca' is not copied to rp_m (line 9 of algorithm 10). However, by lemma 4.6.3, the forward graph of bca' is already contained in rp_m.
Conversely, if $bca' \notin rp_1.SBCA$ then $bca' \notin Path(b',f)$ for any $b' \in B$ and $f \in smp.F$ because every forward reachable state is visited in a projection of b'. Then $\nexists\ bca''.\ (\exists\ path(rp_2.initEBC,bca''),path(bca'',bca') \wedge bca'' \in Path(b',f)$ for any $b' \in B$ and $f \in$

smp.F). Then all states and transitions visited on p between rp_2.initEBC and bca' are copied to rp_m. $\square$

Part 2: *rp_m produces no trace that is neither contained in rp_1 nor rp_2.*
For part 2, assume the opposite. Then, there would have to be some BCA or transition in rp_m that adds to a new trace and that is neither contained in rp_1 nor in rp_2. During copying rp_1 (line 1), no new states or transitions are created. As long as no boundary node is encountered, copying transitions and states of rp_2 just reproduces rp_2. For a boundary node b, however, Path(rp_2.initEBC,b) is retained without structural modification in rp_m and $Trace^{trail}_{rp_m}(b) = Trace^{trail}_{rp_2}(b)$ by means of lemma 4.6.3 $\square$

Lemma 4.6.2 (Path Independent State Mapping) *Consider some state s $\in$ smp.s_0 $\cup$ smp.SBCA $\cup$ smp.F of a valid SeqMP choreography smp. Then, its mapping to a state s' $\in$ rp.S of some role projection rp of smp created according to the projection algorithm (algorithm objects 8 and 9) does not depend on the path on which s is discovered.* $\square$

Proof 4.6.2
When creating a projection rp of SeqMP choreography smp for role r, the following types of transitions have to be considered. Thereby, note that upon processing transitions of cases 2-7, the source state already has been mapped.
Case 1: *t(smp.s_0,"true",bca), bca $\in$ smp.SBCA*
There is no path that starts before smp.s_0, so the claim holds for smp.s_0 and bca.
Case 2: *t(bca1,t#2,bca2), bca1,bca2 $\in$ smp.SBCA $\wedge$ r $\in$ RA(bca1) $\wedge$ r $\in$ RA(bca2)*
If bca2 has not been mapped before, then a copy of bca2 is created. Otherwise, its copy is retrieved (cf. algorithm objects 8 and 9). So, the mapping of bca2 and of t is unique.
Case 3: *t(bca1,t#2,f), bca1 $\in$ smp.SBCA $\wedge$ f $\in$ smp.F $\wedge$ r $\in$ RA(bca1)*
By analogy with case 2.
Case 4: *t(bca1,t#2,bca2), bca1,bca2 $\in$ smp.SBCA $\wedge$ r $\in$ RA(bca1) $\wedge$ r $\notin$ RA(bca2)*
Assume bca2 is not reachable via a second path. Then an event-based choice is created for bca2. Otherwise, if bca2 has not yet been mapped then a dedicated new event-based choice is created as well. If bca2 already has been mapped then by line 22 of algorithm 8 a dedicated new event-based choice has been created for bca2 as well. Transition t then is created between the mapping of bca1 and the event-based choice. So, the mapping of bca2 and t is unique.
Case 5: *t(bca1,t#2,bca2), bca1,bca2 $\in$ smp.SBCA $\wedge$ r $\notin$ RA(bca1) $\wedge$ r $\in$ RA(bca2)*
If bca2 has not been mapped before, then a copy of bca2 is created. Otherwise, its copy is retrieved and a transition between the event-based choice for abstracting bca1 and the copy of bca2 with t#2 as guard is created. So, the mapping of bca2 and t is unique.
Case 6: *t(bca1,t#2,f), bca1 $\in$ smp.SBCA $\wedge$ f $\in$ smp.F $\wedge$ r $\notin$ RA(bca1)*
By analogy with case 5.
Case 7: *t(bca1,t#2,bca2), bca1,bca2 $\in$ smp.SBCA $\wedge$ r $\notin$ RA(bca1) $\wedge$ r $\notin$ RA(bca2)*
Assume, bca2 is not reachable via a second path. Then bca2 is mapped to the event-based choice used for abstracting bca1, t is not mapped, and the mapping is unique.

Otherwise, if bca2 has not yet been mapped then by line 22 of algorithm 8 a dedicated new event-based choice is created. If it already has been mapped a dedicated new event-based choice has been created for bca2 by the same argument. So, the mapping of bca2 and t is unique. □

Lemma 4.6.3 (Isomorphic Subgraph Mappings) *Consider some state $s \in smp.s_0 \cup smp.SBCA \cup smp.F$ of a valid SeqMP choreography smp. Then the graph that results from mapping the forward reachable subgraph of s during some role projection rp of smp is isomorphic for all paths on which s may be discovered.* □

Proof 4.6.3
Iteratively apply lemma 4.6.2 during a forward traversal of the graph. □

Lemma 4.6.4 (Unique Projection Start Pattern)
Consider multiple projections $rp_1,...,rp_n$ of a valid SeqMP choreography smp for the same role r. Then, all of these projections rp_i start with the initial state linking only to an event-based choice:
$rp_i.s_0 = smp.s_0 \wedge (\exists\ t(s_0,\text{"true"},e) \in rp_i.T.\ e \in rp_i.EBC \wedge (\nexists t' \in rp_i.T.\ t' \neq t_1 \wedge t'\#1 = rp_i.s_0))$. □

Proof 4.6.4
If there are multiple projections $rp_1,...,rp_n$ for role r then r does not participate in the first BCA of smp and hence $smp.s_0$ is connected to an event-based choice as of algorithm 9. Otherwise, there would be only a single projection as every state of a valid SeqMP choreography is reachable via the first BCA. □

While all BCAs without participation of the focal role are abstracted away by means of the projection algorithm, the representation of projections still can be optimized. Figure 4.20 shows one out of two possible projections for the FSP role of the use case depicted in figure 4.15. It contains only one BCA, but five EBCs and thirteen final states are included to describe permissible behavior. The following five reduction rules have been identified to simplify projections. Rules 4.6.1, 4.6.2 and 4.6.3 do not affect the set of execution traces in the strict formal sense. Rules 4.6.4 and 4.6.5 do affect the set of execution traces because final states are removed from the projections and final states are accounted for in execution traces. However, this can be justified by the following consideration: If an EBC links to multiple final states then the focal role is not aware of which final state is reached at runtime (without notification outside the choreography definition). So, the focal role just has the information that the process may be terminated, but it does not know when and with which result. As a consequence, all the focal role needs to know is that the process may be terminated after an event-based choice and hence multiple final states after an EBC can be conflated.

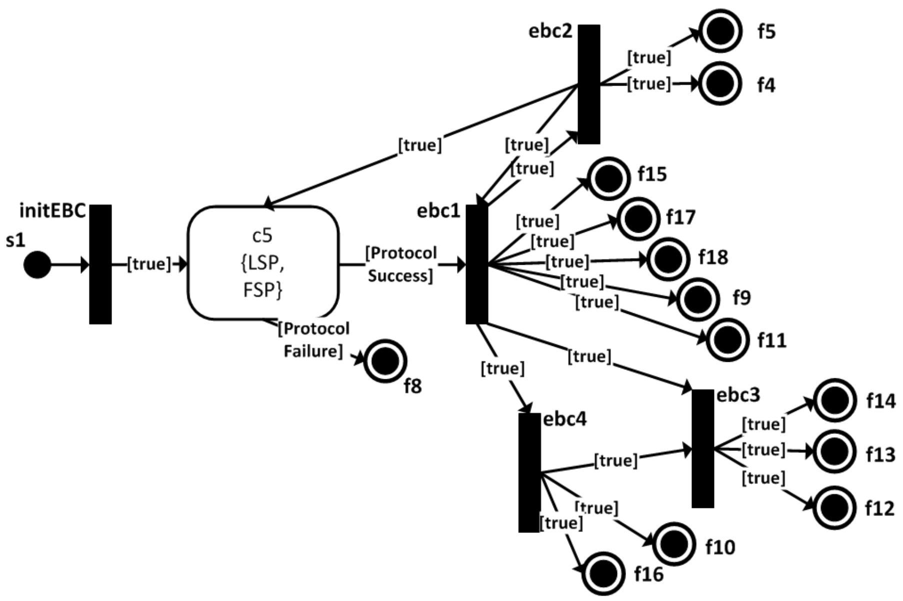


Figure 4.20.: 1 out of 2 Possible Projections for the FSP Role (Use Case of Figure 4.15)

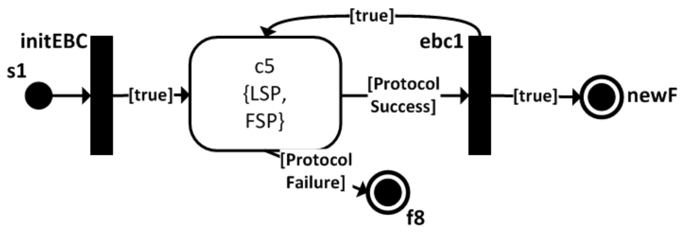


Figure 4.21.: Projection of Figure 4.20 after Applying Reduction Rules

By applying the following reduction rules the projection depicted in figure 4.20 with five EBCs and thirteen final states can be reduced to the projection depicted in figure 4.21 with only two EBCs and two final states.

Rule 4.6.1 (Subsequent Event-Based Choices)
Let ebc_1, ebc_2 be event-based choices of a role projection rp of some role r,
$T^{inter} = \{t \in rp.T \mid (t\#1 = ebc_1 \land t\#3 = ebc_2) \lor (t\#1 = ebc_2 \land t\#3 = ebc_1)\}$,
$T^{fwd}_{ebc_1} = \{t \in rp.T \mid t\#1 = ebc_1\} \setminus T^{inter}$,
$T^{fwd}_{ebc_2} = \{t \in rp.T \mid t\#1 = ebc_2\} \setminus T^{inter}$ *such that*
$\nexists\, t \in rp.T \setminus T^{inter}.\ t\#3 = ebc_2$.
Then, rp can be simplified without affecting the execution traces of r by performing the following modifications in order (to be interpreted as assignments, not equations):

1. $rp.T = rp.T \setminus T^{inter}$
2. $\forall\, t \in T^{fwd}_{ebc_2}$:
   $rp.T = rp.T \cup \{(ebc_1, t\#2, t\#3)\}$

3. $rp.T = rp.T \setminus T^{fwd}_{ebc_2}$

4. $rp.EBC = rp.EBC \setminus ebc_2$

This reduction rule can be applied to conflate ebc1 and ebc2 of figure 4.20. The informal argument for its correctness is that firing transitions between ebc1 and ebc2 is transparent to the user.

Rule 4.6.2 (Multiple Event-Based Choices)
Let $MEBC = ebc_1,...,ebc_n$ *be a set of event-based choices of a role projection rp of some role r indexed by* $I = [1;n]$,
$T^{inter} = \{t \in rp.T \mid t\#1 = ebc_i \wedge t\#3 = ebc_j \wedge i,j \in I \wedge i{\neq}j\}$,
$T^{fwd}_{ebc_i} = \{t \in rp.T \mid t\#1 = ebc_i\} \setminus T^{inter}$ *for* $i \in I$ *such that*
$(\forall\ i \in [2;n].\ \exists\ t \in T^{inter}.\ t\#1 = ebc_1 \wedge t\#3 = ebc_i) \wedge$
$(\forall\ i \in [2;n].\ \nexists\ t \in rp.T \setminus T^{inter}.\ t\#3 = ebc_i)$.
Then, rp can be simplified without affecting the execution traces of r by performing the following modifications in order (to be interpreted as assignments, not equations):

1. $rp.T = rp.T \setminus T^{inter}$

2. $\forall\ t \in \bigcup_{i\in[2;n]} T^{fwd}_{ebc_i}$:
   $rp.T = rp.T \cup \{(ebc_1,t\#2,t\#3)\}$

3. $rp.T = rp.T \setminus \bigcup_{i\in[2;n]} T^{fwd}_{ebc_i}$

4. $rp.EBC = rp.EBC \setminus (MEBC \setminus \{ebc_1\})$

This reduction rule can be applied to conflate ebc1, ebc3 and ebc4 of figure 4.20 and the argument for its correctness is analogous to rule 4.6.1. Strictly speaking this rule is the generalized form of rule 4.6.1.

Rule 4.6.3 (Loop)
Let bca be a BCA and $MEBC = ebc_1,...,ebc_n$ *be a set of event-based choices of a role projection rp of some role r indexed by* $I = [1;n]$, $PRED_{bca} = \{t \in rp.T \mid t\#3 = bca\}$,
$T^{inter} = \{t \in rp.T \mid t\#1 = s_1 \wedge t\#3 = s_2 \wedge s_1,\ s_2 \in MEBC \cup \{bca\} \wedge s_1 \neq s_2\}$,
$T^{fwd}_{ebc_i} = \{t \in rp.T \mid t\#1 = ebc_i\} \setminus T^{inter}$ *for* $i \in I$ *such that*
$(\forall\ i \in [2;n].\ \exists\ t \in T^{inter}.\ t\#1 = ebc_1 \wedge t\#3 = ebc_i) \wedge$
$(\forall\ i \in [2;n].\ \exists\ t \in T^{inter}.\ t\#1 = ebc_i \wedge t\#3 = ebc_1) \wedge$
$(\forall\ i \in [2;n].\ \exists\ t \in T^{inter}.\ t\#1 = bca \wedge t\#3 = ebc_i) \wedge$
$(\exists\ t \in rp.T.\ t\#1 = ebc_1 \wedge t\#3 = bca) \wedge$
$(\forall\ i \in [2;n].\ \nexists\ t \in rp.T \setminus T^{inter}.\ t\#3 = ebc_i)$.
Then, rp can be simplified without affecting the execution traces of r by performing the following modifications in order (to be interpreted as assignments, not equations):

1. $rp.T = rp.T \setminus ((T^{inter} \setminus PRED_{bca}) \setminus SUCC_{bca})$

2. $\forall\ t \in \bigcup_{i\in[2;n]}\ T^{fwd}_{ebc_i}$:
   $rp.T = rp.T \cup \{(ebc_1,t\#2,t\#3)\}$

3. $\forall\ t \in (SUCC_{bca} \cap T^{inter})$
   $rp.T = rp.T \cup \{(t\#1,t\#2,ebc_1)\}$

4. $rp.T = rp.T \setminus \bigcup_{i\in[2;n]}\ T^{fwd}_{ebc_i}$

5. $rp.T = rp.T \setminus (SUCC_{bca} \cap (T^{inter} \setminus \{t \in T^{inter} \mid t\#3 = ebc_1\}))$

6. $rp.EBC = rp.EBC \setminus (MEBC \setminus \{ebc_1\})$

This rule is not reflected in the projections of figures 4.18, 4.19, 4.20.

Rule 4.6.4 (Multiple Final States)
Let ebc be an event-based choice of a role projection rp of some role r, T^f a set of transitions such that $\forall\ t \in T^f.\ t\#1 = ebc \wedge t\#3 \in rp.F \wedge (\nexists\ t' \in rp.T.\ t' \neq t \wedge t'\#3 = t\#3$, and T^{alt} a non-empty set of transitions such that $\forall\ t \in T^{alt}.\ t\#1 = ebc \wedge t\#3 \notin rp.F$.
Then, rp can be simplified by performing the following modifications in order (to be interpreted as assignments, not equations):

1. $rp.T = rp.T \setminus\ T^f$

2. $rp.F = rp.F \setminus \{f \mid \exists\ t \in T^f.\ t\#3 = f\}$

3. $rp.F = rp.F \cup f$, *f a new FinalState*

4. $rp.T = rp.T \cup \{(ebc, \text{"true"}, f)\}$

This reduction rule can be applied to conflate all final states of figure 4.20 connected to ebc1, ebc2, ebc3 and ebc4 after having applied rules 4.6.2 and 4.6.1. The rationale for doing so has been explained above.

Rule 4.6.5 (Event-Based Choice and Final States) *Let ebc be an event-based choice of a role projection rp of some role r, $PRED_{ebc} = \{t \in rp.T \mid t\#3 = ebc\}$, $T^{final} = \{t \in rp.T \mid t\#1 = ebc \wedge t\#3 \in rp.F\}$ such that $\nexists\ t'$ in $rp.T \setminus\ T^{final}.\ t'\#1 = ebc \vee t'\#3 = t\#3$ for some $t \in T^{final}$.*
Then, rp can be simplified by performing the following modifications in order (to be interpreted as assignments, not equations):

1. $rp.T = rp.T \setminus\ T^{final}$

2. $rp.F = rp.F \setminus \{s \mid \exists\ t \in T^{final}.\ t\#3 = s\}$

3. $rp.F = rp.F \cup f$, *f a new FinalState*

4. $\forall$ *t* $\in$ *$PRED_{ebc}$:*
$rp.T = rp.T \cup \{(t\#1,t\#2,f)\}$,
$rp.T = rp.T \setminus \{t\}$

5. $rp.EBC = rp.EBC \setminus \{ebc\}$

This rule can be applied to conflate ebc1, f12, f13, and f14 of figure 4.19.

4.7. Chapter Summary

This chapter introduced CHORCH's choreography styles for B2Bi choreography modeling. The definition of these styles, in particular the definition of ebBP-Reg and ebBP-ST, is based on the ebBP format. ebBP is used as choreography format for the reasons described in chapter 3. The two most important reasons are the suitability of ebBP as B2Bi choreography exchange format (due to its XML syntax) and the availability of B2Bi domain concepts like BTs that allow the modeler to leverage *common knowledge*.

Yet, there are some deficiencies in using ebBP *as is* for the purpose of choreography modeling that provide the motivation for the definition of CHORCH's B2Bi choreography styles:

- **Amendments of ebBP** Section 4.1 describes a series of shortcomings in the ebBP specification. Most notably, the exchange procedures that are supposed to implement ebBP BTs allow for inconsistent results of the integration partners. This thesis provides an improvement in that regard by providing an BT execution model in section 4.3 that is formally defined, has formal execution semantics, is formally validated by means of model checking techniques (see appendix D) and is implementable using Web Services and BPEL technology (see next chapter).

- **Unambiguous execution semantics** ebBP does not define a formal execution semantics for B2Bi choreography models. As the definition of an execution semantics (if implementability is a requirement) is necessarily affected by the available primitives of the execution environment, this is not a very surprising fact. ebBP is deliberately technology-agnostic and therefore limited in making assumptions about the execution environment.

  CHORCH defines ebBP-Reg, ebBP-ST and SeqMP as choreography styles that are aligned with specific modeling requirements (cf. chapter 3). This styles are formally defined and provided with a formal operational execution semantics. While the execution semantics for ebBP-Reg and ebBP-ST is defined explicitly, the SeqMP execution semantics is simply defined to comply with state machine semantics. This is acceptable as SeqMP is not conceived as implementation contract, but rather as a model for the analysis of multi-party choreographies where the crucial information is the sequence of component BCAs. The formal execution semantics of ebBP-Reg and ebBP-ST is the precondition for unambiguous interpretation of choreography models and the development of tooling for modeling, analysis and translation into implementation artifacts.

Note that the defined execution semantics is applicable to CHORCH's choreography styles only and not to arbitrary ebBP models. As CHORCH's choreography styles are known to cover the majority of integration scenarios for the identified type of B2Bi (cf. chapter 3), this is a minor limitation.

- **Characterization of model validity** The characterization of model validity requires the definition of the term *"validity"*. CHORCH defines a B2Bi choreography model to be valid if and only if it is an instance of either ebBP-Reg, ebBP-ST or SeqMP. Validity in this sense implies that a corresponding model can either be translated into a Web Services and BPEL-based implementation (see next chapter) or that it can be analyzed using the SeqMP framework. The rules for defining valid models are given in sections 4.4.3, 4.5.2 and 4.6.1.

  As ebBP does neither define concrete choreography styles that are aligned with specific modeling requirements nor formal execution semantics, an advanced characterization of model validity as provided by CHORCH is missing.

Beyond these benefits it is worth noting that CHORCH's choreography styles are both *simple* and *sufficient*.

Simplicity of CHORCH's B2Bi choreography styles is fostered by using state machines as underlying paradigm for the specification. This enables an intuitive modeling approach where the user is allowed to connect interaction activities in an almost unconstrained way. In particular, there is no need to think in structured control flow definitions [76] that heavily constrain modeling. Furthermore, the execution semantics of ebBP-ST, ebBP-Reg and SeqMP as well as the corresponding BPEL implementation and analysis framework ensure that the global perspective on choreographies is also retained for the local views of the integration partners. This is different from other approaches such as [28, 33, 232] where the local view of integration partners on the actual message exchanges may deviate from the global choreography definition due to the use of asynchronous messaging facilities.

Sufficiency of CHORCH's B2Bi choreography styles is given by means of alignment with the requirements derived from this work's underlying literature review and a significant set of real-world use cases (cf. chapter 3).

The validation of CHORCH's B2Bi choreography styles is provided in several ways. Firstly, the formal definition of each ebBP dialect together with the definition of formal execution semantics as well as modeling concrete use cases in ebBP allows for reciprocal detection of design and formalization errors. Secondly, the sanity of the SeqMP model is validated using a prototypic implementation of the analysis algorithms and by proving the correctness of the SeqMP role projection algorithm. Thirdly, the implementability of ebBP-ST and ebBP-Reg on top of the integration architecture introduced in section 4.2 using Web Services and BPEL technology is given.

As the binary choreography styles are supposed to be used as implementation contract between integration partners, the derivation of corresponding implementation artifacts is of particular interest. The next chapter will therefore provide a more detailed description of the integration architecture, spell out the technology assumptions made about the environment, and describe the BPEL mapping rules for the corresponding ebBP models.

5. Implementation of Choreographies as BPEL Orchestrations

This chapter shows how the execution model for ebBP BTs as well as ebBP-Reg can be implemented using Web Services and BPEL technology and thus demonstrates the implementability of the corresponding execution semantics. As a by-product, guidelines for automatically generating BPEL-based implementations are presented. However, note that one valid way of implementing the semantics is described and not the only valid way. By choosing ebBP-Reg instead of ebBP-ST the more demanding model in terms of control flow is presented. Moreover, the mapping of ebBP-ST to BPEL is covered in [182] and given in appendix B so that the mapping of the binary choreography styles of the previous chapter can be considered to be complete. SeqMP is not covered in this chapter because it is designed as a framework for the analysis of multi-party choreographies and not as an abstract implementation specification.

For the discussion of the implementation, remind that the purpose of B2Bi choreographies is the specification of the message exchanges between the BSIs of the interacting parties in order to facilitate interoperable implementation. Therefore, a description of B2Bi system implementation requires a precise description of the message send and receive events of each individual interaction partner. In the following, using public orchestrations, private orchestrations or so-called *control processes* for this purpose is discussed.

Non-executable public orchestration processes (cf. section 2.2) could be specified that have to be refined with business logic later on in order to derive fully executable processes. Such a public orchestration approach is disadvantageous in several ways. Firstly, implementability of public orchestrations and the effort needed for implementation is unknown. Secondly, depending on the public orchestration language, refining public orchestrations may be hard if not impossible from a practical point of view. During the course of this thesis, prototypic BPEL-based implementations of BTs and BCs comprised hundreds to thousands of lines of BPEL code. If auto-generated, manually editing such BPEL definitions is cumbersome and error-prone. Thirdly, refining public orchestrations may impede compliance of the BSI implementation to the choreography definition due to unintended control flow modifications when filling in business logic.

Alternatively, the description of the message send and receive events of each individual interaction partner could be given by means of complete implementations of BSIs in the sense of private orchestrations (cf. section 2.2). However, such an approach limits reuse of orchestration definitions as these are tailored to one specific environment. Moreover, automatic generation of implementations is impeded if complete private orchestrations need to be defined. Firstly, taking the heavily varying IT systems in the B2Bi domain into account of a model-driven approach adds tremendous complexity. Secondly, integration partners or their IT partners may be reluctant to adopt automatically generated private orchestrations because the interleaving of business document exchanges with internal processes may be a source of (perceived) competitive advantage. Finally, giving complete private orchestration definitions blurs the boundaries between control flow logic and private business logic so that it is not clear what kind of modifications are admissible without affecting protocol interoperability.

Thus, this work proposes specific *control processes* as a hybrid approach for characterizing the admissible message send and receive actions of each integration partner. Control processes are fully executable orchestrations that implement the control flow of choreography definitions and import business logic by means of standardized interfaces that are derived from the choreography definition. Control processes are like public orchestrations in the sense of unambiguously capturing the admissible public message exchange sequences of an integration partner and not specifying private logic in full detail. At the same time, control processes are like private orchestrations in the sense of being fully executable. For better understanding the concept of control processes, reconsider the basic integration architecture as depicted in figure 4.1 on page 88 that describes the interaction between two integration partners. Furthermore, assume that the interactions are supposed to follow a predefined choreography specification. The application logic of both partners such as detecting the need for a new collaboration instance, the creation of business documents or the validation of business documents is encapsulated in the business applications or backend systems depicted as white boxes. Conversely, the shaded boxes depict the control processes that ensure adherence of the message exchanges to the choreography specification on the one hand and agreement upon the result of message exchanges on the other. In order to ensure adherence to the choreography specification, control processes associate messages with process instances, check whether incoming or outgoing messages are admissible at a particular point in time, and technically coordinate the selection of one out of many concurrent events if need be. For example, if each integration partner tries to trigger a separate BTA, then the control processes make sure that these are not executed at the same time. Figure 4.1 shows multiple gray boxes for representing control processes due to the fact that a modularized structure of control processes is proposed where a separate pair (one for each interaction partner) is used for each BTA or BCA of a BC. In how far different systems for providing backend functionality are used and, in consequence, whether or not multiple white boxes are adequate for representing backends in figure 4.1 is actually left to the integration partners. The details of message exchanges between control processes of interacting parties on the one hand and control processes and backend systems of the same partner on the other are given in the next sections. For comparison to the pubic orchestration approach and the private orchestration approach, note the following advantages of the control process approach that result from separating business logic and control flow logic:

- The definition of control processes is fully executable. Hence, cumbersome and error-prone modification of process definitions once these have been derived from choreography definitions is not needed.
- Control processes ensure strict compliance to the choreography definition and as there is no need for modifying control processes, strict compliance is not endangered. Note that local optimizations are possible nonetheless. For example, if an integration partner has the option to decide between two alternative BTAs at a particular point of the choreography definition, then the backend may choose to always trigger the same BTA. Moreover, control processes provide a protocol specification that completely is in the domain of one integration partner. So, if local optimizations are to be analyzed, these can be tested against the local control process and not against the partner's remote BSI. This basically amounts to moving the protocol interoperability problem into the "safe" environment of one partner.
- The automation of orchestration derivation compared to complete private orchestrations is simple because a set of standard interfaces can be used for abstracting away business logic.
- Control process definitions can be reused more easily as standard interfaces are used for accessing business logic. This basically means that adapting a control process definition to a different IT environment can be done by means of providing different implementations of the backend interfaces.
- Control processes can be used as reference implementation. Performance optimizations and reuse of existing implementations can be valid reasons for adopting proprietary BSI implementations instead of auto-generated control processes. Even if integration partners are not willing to implement their interactions by means of auto-generated control processes, then these can still be used as reference for the actual BSI implementations. As control processes are executable, these could be used for generating sample message exchange sequences. Of course, some dummy implementations of backend interfaces would be needed for that.

The downside of using standardized control process implementations is performance impact. The association of messages with process instances upon receipt of messages, coordination of control flow paths as well as checks for the admissibility of particular message types takes some time. Contrarily, traditional B2Bi gateways tend to just treat messages as process-agnostic content and relay these to some business applications that will do the process association. In consequence, traditional gateway technology (beyond having matured for decades) should by far outperform control process based BSI implementations in terms of throughput and scalability. However, this work's focus of mapping B2Bi choreographies to control processes clearly is a matter of improving functionality instead of performance optimization and a detailed analysis of the performance impact of adopting control processes therefore is not provided.

Web Services and BPEL are selected as technological basis for implementing control processes. The heterogeneous setting of B2Bi scenarios clearly calls for a dedicated interface technology such as Web Services that is able to bridge different computing platforms and programming languages. Moreover, Web Services are a natural candidate for wrapping existing IT systems and therefore promise easy provision of standardized interfaces for accessing business applications (as required for control processes). Finally, Web Services offer the potential of cost gains by using open Internet protocols instead of VANs and make related XML-based technologies accessible (cf. sections 1.2 and 2.1). In particular, BPEL as standardized orchestration language relies on the use of Web Services. While basic Web Services are used for bridging the heterogeneous systems of integration partners and for wrapping existing systems, BPEL is used for implementing control processes as orchestrations. Obviously, control processes not necessarily have to be implemented using BPEL. Proprietary B2Bi frameworks or plain enterprise computing platforms such as Microsoft's .NET[1] or the Java Enterprise Edition[2] may be convenient depending on the IT landscape of the interacting parties. The list below motivates why a standardized orchestration language should be used nonetheless. The arguments reflect the availability of an explicit process description format that offers constructs at the orchestration level such as capturing message send and receive events. Hence, proprietary B2Bi frameworks that offer a corresponding format offer those advantages as well. However, note that proprietary formats tend to be disadvantageous in terms of portability and open accessibility.

1. **Easy reuse of process definitions.** BPEL process definitions can directly be deployed onto BPEL engines so that reuse of control process implementations basically amounts to copying the BPEL process definitions together with the corresponding WSDL interfaces. In theory, this also holds true for porting process definitions to process engines of different vendors. In practice, the use of proprietary extensions and the non-uniform implementation of BPEL features may significantly impact portability. However, porting a process definition to a different engine that has been written for the standard still promises to be easier than porting an application between two independently developed frameworks.

2. **Use of standard tools for logging, monitoring, testing and management.** Several implementations of BPEL such as IBM's Business Process

[1]`http://www.microsoft.com/net/`, last access: 12/20/2011
[2]`http://jcp.org/en/jsr/detail?id=316`, last access: 12/20/2011

Manager[3], Oracle's BPEL Process Manager[4] or Apache ODE[5] are available. Moreover, the open accessibility of BPEL fosters the development of independent solutions for logging, monitoring, testing and management in general. Note that a BPEL process essentially can be interpreted as a Web service so that generic Web Services tools can be used for managing BPEL processes as well. For example, soapUI[6] can be used for sending test messages to BPEL processes or for providing mock-up services.
Of course, management tools are available for enterprise computing platforms as well, but these typically are not tailored to the orchestration level as required for the management of control processes.

3. **Conformance checks and analysis.** BPEL enables an explicit representation of control flow which is the basis for checking conformance of control process implementations to the agreed-upon choreography definitions. Preliminary results for testing the control flow conformance of control processes to choreography definitions by means of model checking are available in [45, 46]. Apart from that, explicit process definition as provided by BPEL is the basis for a wealth of (frequently scientific) approaches for generating, analyzing, reorganizing, visualizing and managing orchestrations.

In recent time, Web Services technology and to some extent BPEL are blamed for being overly complex and not living up to the promises that have been made upon their invention. Indeed, there are issues concerning interoperability, performance, portability and complexity.

Regarding Web Services, the fact that the WS-I profiles significantly constrain the WSDL, SOAP and other related Web Services standards (cf. section 2.1) shows that some functionality of those standards is either superfluous or not captured precisely enough. The results of the next section will acknowledge this fact by demonstrating that interoperable implementation of WS-* functionality by WS stacks cannot be relied upon. Moreover, it is not always clear how using the wealth of SOAP functionality is supposed to be asserted at the WSDL level which challenges the concept of declaratively specifying the message exchanges between service consumer and service provider. Moreover, the DOM-based implementation of some Web Services and XML libraries impedes the exchange of large payloads due to memory constraints. However, the size of business documents may amount to several hundred megabytes which rules out DOM-based processing in practice.

Regarding BPEL, homogeneous support for BPEL features cannot be assumed. For example, the openESB BPEL engine used in this work does not support control links between the activities of a BPEL flow construct or the parallel execution of the

[3] `http://www-01.ibm.com/software/integration/business-process-manager/`, last access: 12/20/2011

[4] `http://www.oracle.com/technetwork/middleware/bpel/overview/index.html`, last access: 12/20/2011

[5] `http://ode.apache.org/`, last access: 12/20/2011

[6] `http://www.soapui.org/`, last access: 12/20/2011

BPEL foreach construct. Conversely, it offers extensions for logging, date and time manipulation as well as error handling that go beyond standard BPEL functionality. In practice, this is problematic when porting BPEL applications to different engines although this still can be done as shown by the port of BT implementations to the Oracle BPEL Process Manager for the approach in [54]. Apart from that, BPEL is blamed to be a rather bulky format. On the one hand, this critique is not always justified because the orchestration of XML messages that may stem from different domains in a standardized format bears inherent complexity. On the other hand, even simple tasks such as variable assignments may require several XML tags and it is hard to create syntactically valid BPEL processes without considerable tooling support.

While the criticism of Web Services and BPEL technology is justified to some extent the question for alternatives cannot be answered satisfactorily if the aspects of interoperability, cost, process awareness, visibility and reuse are taken into account.

Traditional B2Bi technologies such as EDI or AS2 solve the problem of transmitting large messages with high performance, but typically also require more costly implementations and leave the problem of controlling conformance to choreographies to the business applications. However, the modular integration architecture proposed for this work conceptually supports using traditional B2Bi technologies for performing business document exchanges while using Web Services technology for controlling BCAs. A concept for how this can be achieved has been published in [188].

Representational State Transfer (REST)-style interactions [42] frequently are mentioned as alternative style for Web Services based interactions. However, REST implies a contract-less way of designing interactions between some user agent and a server in which the user agent discovers valid ways of interacting with the server from the content provided by the server. A predefined set of URLs as convention for interfaces that need to be called in a predefined sequence (for implementing a protocol) actually contradicts the REST paradigm. The format of the delivered content must comply with some conventions so that it can be rendered by the user agent, but predefined semantics of content are not foreseen by REST. In the end, some kind of intelligence which is typically provided by a human user must be available for making sense of the transmitted content. Moreover, *"the application state is controlled and stored by the user agent and can be composed of representations from multiple servers"* [42]. This contradicts the concept of a shared protocol state of B2Bi partners that reflects the series of business documents that already have been exchanged in the course of a process instance. In summary, a truly REST-based realization of a bidirectional protocol-based implementation of a B2Bi choreography definition is inadequate. REST is an architectural paradigm that describes the scalable and flexible design of web application architectures and not a framework for enterprise computing. This judgment is in line with the inventor of REST who states that *"The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction"* [42] as well as [158] who advise *"[..]to use RESTful services for tactical, ad hoc integration over the Web* *(à la Mashup) and to prefer WS-* Web services in professional enterprise application integration scenarios with a longer lifespan and advanced QoS requirements".*

Finally, the question why the ebXML framework technologies ebMS and CPPA are not used must be answered. While ebMS is an alternative communication technology that could be used instead of Web Services, CPPA is a format for specifying the technical details of interactions such as transport protocols or endpoints. The ebMS versions 2.0 [129] as well as 3.0 [136] both are based on SOAP (as much as Web Services) but do not use WSDL for providing an interface for interactions. Hence, while the performance issues of SOAP-based interactions would remain when using ebMS, the power of generating artifacts and protocol stubs from WSDL interfaces by means of standard WS stacks would be lost. Similarly, the lack of an interface definition format would impede the use of BPEL that relies on the explicit description of interfaces. Moreover, ebMS libraries are by far not as commonly available as WS stack implementations. Consistently, ebMS is the only technology of RosettaNet's MMS[7] program that has not been validated so far.
Instead of using CPPA for the configuration of transport level details, the B2Bi choreography styles of this work are mapped to BPEL and WSDL assuming standard values for the WSDL protocol binding. The core reason for this is that the BPEL mapping is supposed to demonstrate the feasibility of implementing this work's B2Bi choreography styles and not to define a comprehensively configurable implementation framework. Moreover, WSDL separates the protocol binding part from the interface definition part so that other protocol configurations can replace the existing definitions relatively easy. In addition, the current version 2.0 [128] of CPPA is tailored to ebMS and requires considerable rework to be amenable to alternative communication technologies. Version 3.0 [127] of CPPA adds the so-called *WSSenderBinding* and *ediintSenderBinding* for leveraging Web Services and ediint[8] as alternative communication technologies to ebMS. However, this version of CPPA is still in an incomplete state and the latest draft is more than two years old.

In summary, Web Services and BPEL technology may not represent the optimal technology mix for implementing B2Bi interactions. But, if interoperability, cost, process awareness, visibility and reuse are of major importance then there is barely an alternative.

The chapter proceeds as follows. Section 5.1 discusses the general aspects of the implementation such as the realization of advanced security and reliability features based on Web Services. Section 5.2 describes the BPEL-based implementation of BTs and section 5.3 describes the BPEL-based implementation of BCs. The feasibility of automatically deriving such implementations has been shown in [54] and in [182].

[7]`http://www.rosettanet.org/Standards/RosettaNetStandards/MultipleMessagingServices/tabid/474/Default.aspx`, last access: 12/20/2011
[8]`http://www.ietf.org/wg/concluded/ediint.html`, last access: 12/20/2011

5.1. General Implementation Aspects

The general aspects of implementing this work's B2Bi choreography styles concern the underlying WS stack and BPEL engine, the identification of interacting software components, message header and process correlation information, the combination of synchronous and asynchronous coordination styles, and the availability of advanced security and reliability features.

Implementation Platform As implementation platform, the Integrated Development Environment (IDE) NetBeans 6.7.1, the application server GlassFish 2.1.1 and the enterprise service bus openESB 2.6.1 available in an all-in-one distributable as GlassFish ESB bundle 2.2 have been used. While openESB provides, among others, a BPEL engine and an XSLT library, GlassFish provides Metro as an advanced Web Services stack and EJB as well as servlet technology for Web Services implementation. Moreover, NetBeans provides a comfortable interface for developing, testing and debugging all relevant implementation artifacts. At the time selected, this combination outperformed alternative technology combinations either in terms of functionality or usability. IBM's WebSphere platform as well as Apache's Axis2 stack do not provide sufficient support for WS-Policy based security and reliability realization (cf. appendix C and [16]). In turn, Oracle's Business Process Manager provides limited debugging and logging support for BPEL process definitions. However, the choice of implementation platform is of minor importance as long as the core artifacts of the implementation concept can be tested which is the case for the selected platform. Since Oracle acquired SUN, the combination of NetBeans, GlassFish and openESB is not provided any more. While Oracle explicitly announced further support for NetBeans and GlassFish in a white paper[9], the future of openESB has not been clarified at all and its fate is left to the open source community. As the choice of implementation platform is of minor importance, lacking support for openESB by Oracle does not limit the validity of this work's results. In that regard, note that the implementation of BTs has been ported to Oracle Business Process Manager for the approach in [54].

Identification of Software Components For the identification of interacting software components reconsider the scheme of this work's modularized integration architecture as given in figure 4.1 on page 88. Control processes are used to control and implement the cross-organizational message exchanges of integration partners, but explicitly do not contain application logic which is accessible by means of well-defined interfaces. Furthermore, remember that the implementation is designed for exactly two integration partners. A separate pair of control processes for each BCA and BTA of a BC is used in order to modularize the implementation. Based on this basic structure, the following terminology will be used throughout this chapter.

[9]`http://www.oracle.com/us/038563.pdf`, last access: 12/20/2011

The term *"top-level control process"* refers to one of the BPEL processes of the interaction partners that controls the control flow of the root BC that has been agreed upon by the interaction partners. As the implementation is designed for binary choreographies, there are exactly two top-level control process definitions for a particular root BC. A *"BC control process"* refers to a BPEL process that controls the control flow of a particular type of BCA for one of the interaction partners. As choreographies may be composed hierarchically, several types of BC control processes may exist for a particular partner choreography. Note that the top-level control processes are BC control processes as well. Similarly, the term *"BT control process"* refers to one of the BPEL processes that controls the control flow a particular BT. Again, there are exactly two BT control process definitions per type of BT and as the roles *"requester"* and *"responder"* (cf. section 2.3.1) are statically defined for BTs, the terms *"requester control process"* and *"responder control process"* may be used to refer to the requester's and responder's control process, respectively. Whenever the relationship between a superordinate and subordinate control process is referred to, the terms *"master/parent control process"* and *"child control process"* may be used to refer to the superordinate and subordinate control process, respectively.

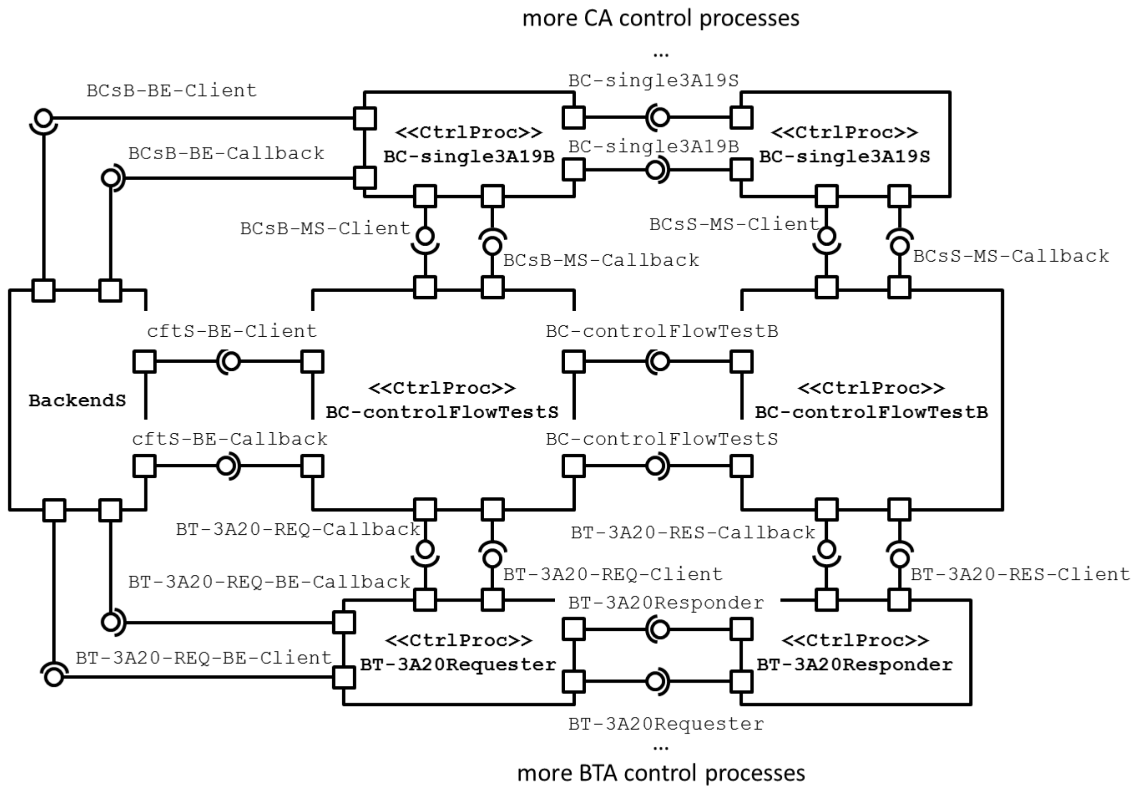


Figure 5.1.: Interfaces for Interacting with a Top-Level Control Process

In addition, *"helper services"* denote any kind of Web services that provide static functionality that does not rely on the process or master data stored in the integration partners' business applications and can easily be provided in the same deployment package as a control process. A typical example for such a helper service is an XPath expression evaluation service for evaluating an XPath expression against a given business document. Finally, the term *"backend"* will be used to refer to any kind of business logic that is provided by existing business applications. Although the structure of those business applications may be quite complex and the backend functionality for a particular BC may be provided by a multitude of systems, the term *backend* will be used as if it would refer to a monolithic block. Thus, the *backend* component abstracts away the complexity of the existing IT landscape.

For each control process a series of interfaces is used for interaction. For each component a control process interacts with, two interfaces are used, one for each direction of communication. The naming of the interfaces follows a simple scheme that takes the types of interacting components into account. The interfaces that a corresponding pair of control processes offer each other just carry the names of the respective control processes, i.e, if a control process carries the name *"X"* then the interface the control process offers its peer is named *"X"* as well. For interaction with the backend, the interface that the control process offers for the backend is named *"X-BE-Client"* and the interfaces that the control process uses for consuming backend functionality is named *"X-BE-Callback"*. Similarly, a control process offers an interface *"X-MS-Client"* to a superordinate master control process and uses the interface *"X-MS-Callback"* for communication to the master control process. Although this kind of naming scheme could also be applied to the interaction between a BC control process and a BT control process, the corresponding interfaces are called *"<nameofBT>-REQ-Client"*, *"<nameofBT>-REQ-Callback"*, *"<nameofBT>-RES-Client"*, and *"<nameofBT>-RES-Callback"* for better readability.

Figure 5.1 shows the set of interfaces that is used for implementing B2Bi choreographies as a UML component diagram. The use case of section 4.5.1 is used as example. Each control process is depicted as a separate component with the stereotype <<CtrlProc>> and for each type of component a control process interacts with the interfaces described above are used for interaction. The seller's control process *BC-controlFlowTestS* (the center component of figure 5.1) implements the top-level collaboration of the use case, i.e., it coordinates the sequence of executing the specified BCAs and BTAs. Using interfaces `cftS-BE-Client` and `cftS-BE-Callback`, the *BC-controlFlowTestS* process receives the trigger for starting the overall collaboration as well as for starting BTAs and it sends back acknowledgements to the backend once the requested BTAs have been started. Before such acknowledgements can be sent to the seller's backend, the *BC-controlFlowTestS* control process informs the buyer's top level control process *BC-controlFlowTestB* that a new BTA is needed, waits until *BC-controlFlowTestB* signals that a new instance of the requested BTA's buyer control process has been created, and then sets a up a new instance of the requested BTA's seller control process. For example, for starting the use case's BTA *BT-3A20*, *BC-controlFlowTestS* would wait until *BackendS* requests this BTA, request a new instance via interface `BC-controlFlowTestB` and await the acknowledgement via `BC-controlFlowTestS`, then inform the backend that the BTA control process has been set up and finally create a new instance of *BT-3A20Requestor*. Note that the

seller's BTA control process indeed is created after informing the backend because the first message exchange between *BT-3A20Requestor* and *BackendS* is initiated by the BTA control process. Conversely, when a new BCA has to be started, e.g., *BC-single3A19-2* of the use case, the top level control processes would first start the BCA's control processes and then inform the backends because the first message exchange between backends and BC control processes is initiated by the backend components.
Figure 5.1 only shows control processes for one type of BTA (BT-3A20) and BCA (BC-single3A19-2), but for any additional type of BTA/BCA additional control process components would have to be installed. Once the pair of control process instances has been created for a new BCA/BTA via the corresponding *-Client interfaces, control is passed on to these control processes which, again, interact with each other for coordinating more deeply nested activities and with the respective backends for including business logic. Once the lower level control processes have performed the work, control is returned to the higher level control processes via the *-Callback interfaces. Note that figure 5.1 does not include the interfaces for interacting with the buyer's backend.

Message Header and Process Correlation Information In this work, a dedicated payload-level message header is used to carry processing information to be used by backends and control processes. In particular, process identification information for automatically routing messages to the intended BPEL instance using BPEL's content-based correlation mechanism is contained. In theory, the SOAP header format as underlying Web Services message format could have been used for this purpose. However, the full power of Web Services technology only is exploited if SOAP messages are generated automatically by WS stacks. Using the SOAP header for carrying the relevant information would imply modifying and processing the exchanged SOAP messages by hand at some point during the WS stack processing cycle which would generate additional effort and impede general applicability of the approach. Note that WS-Addressing [224] does not help in this situation either. WS-Addressing just offers a SOAP-level format for endpoint information, but it does not define any kind of application-specific way for identifying process instances. Although some BPEL engines offer automatic correlation of BPEL instances and use the WS-Addressing format for describing corresponding information, the corresponding correlation mechanisms do not follow any specific standard and therefore tie the solution to the BPEL engine under consideration.

The 'commonMetaBlockType' definition of listing 5.1 shows the information that is used for transporting process identification information. 'RootIdentifier', 'ParentIdentifier', and 'InstanceIdentifier' all are assumed to be globally unique and help in identifying the top-level control process of one interaction partner, the master control process of the control process that receives the message, and the control process that actually receives the message itself. Strictly speaking, the 'ParentIdentifier' and the 'InstanceIdentifier' would be sufficient for enabling BPEL process instance correlation.

The ‘RootIdentifier’ as well as the ‘ProcessDepth’ property that gives the nesting level according to the BC structure are just included in order to optimize retrieval of relevant information by the backend applications. The two additional type definitions of listing 5.1, ‘transactionMetaBlockType’ and ‘collaborationMetaBlockType’, basically just reuse the header information and provide containers for adding additional BT specific or BC specific information. For example, the ‘hasLegalIntent’ as well as ‘isConcurrent’ BT configuration parameters as explained in section 4.3.2.1 are included in the BT header for use by the backend.

Listing 5.1: Control Message Header

```
<xsd:complexType name="commonMetaBlockType">
  <xsd:sequence>
   <xsd:element name="RootIdentifier" type="xsd:string"
     minOccurs="1" maxOccurs="1"/>
   <xsd:element name="ParentIdentifier" type="xsd:string"
     minOccurs="1" maxOccurs="1"/>
   <xsd:element name="InstanceIdentifier" type="xsd:string"
     minOccurs="1" maxOccurs="1"/>
   <xsd:element name="ProcessDepth" type="xsd:int"
     minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="transactionMetaBlockType">
 <xsd:complexContent>
   <xsd:extension base="commonMetaBlockType">
     <xsd:sequence>
       <xsd:element name="hasLegalIntent" type="xsd:boolean"
         minOccurs="1" maxOccurs="1"/>
       <xsd:element name="isConcurrent" type="xsd:boolean"
         minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
   </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="collaborationMetaBlockType">
 <xsd:complexContent>
   <xsd:extension base="commonMetaBlockType">
     <xsd:sequence/>
   </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Listing 5.2 shows the definition of an abstract correlation property for BPEL process instance identification that then has to be mapped to concrete elements of messages such as those of listing 5.1. The element prefix ‘vprop’ is bound to the namespace `http://docs.oasis-open.org/wsbpel/2.0/varprop` as required by the BPEL specification. For easy reuse the correlation properties are defined in a separate WSDL file ‘Correlation-composable.wsdl’ that then is imported by the actual interface WSDL files such as ‘<nameofBT>Requestor.wsdl’ or ‘<nameofBT>-MS-Client.wsdl’ for the purpose of mapping the correlation properties to message elements (by means of `propertyAliases` as shown in listing 5.3). The actual correlation sets are then defined in the BPEL definitions that refer to correlation properties. When actual message exchanges take place, the BPEL engines are then in charge of retrieving concrete values for correlation properties from messages according to the `propertyAlias` mapping information. For this work, a correlation set consists of exactly one string-based correlation identifier.

Listing 5.2: Correlation Properties

```
<vprop:property
 name="prop_CollaborationIdentifier"
 type="xsd:string"/>
```

Listing 5.3 shows three different examples of how the identifier for a control process can be mapped. The `propertyAliases` are taken from a 'BT-requester', a 'BT-MS-Client' and a 'BT-BE-Client' interface. The first property alias shows how to extract a collaboration identifier from an ebBP RA message (the ns prefix is bound to `http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0`) where a concrete RA instance must carry the corresponding identifier in the 'CollaborationIdentifier' field. The attentive reader may have noticed that the header block defined in listing 5.1 is not used for carrying instance identifier information. This just shows that, at the BT level, the header does not necessarily have to be used as long as alternative properties can be used for instance identification. This may be convenient in case some existing BT implementations shall be reused. For new implementations, the use of the defined message header also is desirable for the exchange of business documents and business signals which can easily be accomplished by wrapping the header together with the actual payload within an additional XML tag. The second and third `propertyAlias` of listing 5.3 shows how the message header can be used to extract process instance information from a 'txStartMsg' and a 'DropPip3A20PurchaseOrderConfirmationNotificationMsg' WSDL message. First the WSDL part of the respective WSDL message is identified by means of the `part` attribute of the `propertyAlias` and then the `query` is run against that part. The type of the part is the XML container that wraps the header block as well as the actual payload and the XPath expression 'mb:TransactionMetaBlock/mb:InstanceIdentifier' then retrieves the actual instance identifier from the header information where the prefix 'mb' refers to the header block namespace (which has been set to `urn:rosettanet:specification:interchange:composable:xml:header:1.0` for the RosettaNet example).

Listing 5.3: Sample Property Aliases for Extracting Correlation Information

```
<!-- BT-requester interface -->
<vprop:propertyAlias
 propertyName="corr:prop_CollaborationIdentifier"
 messageType="tns:ReceiptAcknowledgementMsg"
 part="ReceiptAcknowledgementPart">
  <vprop:query>ns:CollaborationIdentifier</vprop:query>
</vprop:propertyAlias>

<!-- BT-MS-Client interface -->
<vprop:propertyAlias
 propertyName="corr:prop_CollaborationIdentifier"
 messageType="tns:txStartMsg"
 part="txStartPart">
  <vprop:query>mb:TransactionMetaBlock/mb:InstanceIdentifier</vprop:query>
</vprop:propertyAlias>

<!-- BT-BE-Client interface -->
<vprop:propertyAlias
 propertyName="corr:prop_CollaborationIdentifier"
 messageType="tns:DropPip3A20PurchaseOrderConfirmationNotificationMsg"
 part="DropPip3A20PurchaseOrderConfirmationNotificationPart">
  <vprop:query>mb:TransactionMetaBlock/mb:InstanceIdentifier</vprop:query>
</vprop:propertyAlias>
```

Combination of Synchronous and Asynchronous Coordination Styles This work combines synchronous and asynchronous coordination between integration partners. Thereby, the terms synchronous/asynchronous *communication* and synchronous/asynchronous *interaction* are used as defined in section 2.1.

In general, communication is synchronous in this work and both communication partners achieve agreement with respect to the success of the message transmission immediately. This agreement is reached by means of messaging level functionality the availability of which is discussed in the next paragraph. This paradigm of immediately agreeing upon success of communication simplifies protocol design enormously because transitions in the BT protocol machines of the communication partners (cf. figures 4.2 and 4.3 in section 4.3) can be fired in lock-step. Similarly, the exchange of control messages needed for coordinating control flow of BCs can be specified more easily.

The alternative to this communication paradigm is asynchronous communication of messages which significantly complicates protocol design because the state of the communication channel then must be respected and agreement protocols are needed for computing a consistent outcome between integration partners. Remember that the BT protocol machines specify the exchange of business signals and business documents within one transaction scope which cannot completely be covered on the messaging level. However, as discussed in section 4.3, the complexity of performing agreement protocols at this level is not acceptable. Note also that the validation of the full BT protocol machines using the SPIN[10] model checker (cf. appendix D) was relatively easy due to the synchronous communication design. Conversely, a conceptual predecessor of this work used protocol machines based on asynchronous communication for BT execution and the analysis of these (which aim at exactly the same purpose) proved to be pretty hard in SPIN in the sense that only a restricted version of the BT protocol machines could be validated (cf. [177, 185]).

Asynchronous interaction is used as paradigm for the generation and processing of business signals or business documents in the BT protocol machines. While the success of message transmission is agreed upon immediately between the communication partners using one messaging level interaction, the processing of messages and the potentially related transmission of subsequent messages is performed in separate steps. This is represented in the BT protocol machines by inserting dedicated states that capture the processing of business content and new transitions for new communication events. For example, the transmission of a business document from BT requester to BT responder is performed as One-Way Web Services call that corresponds to one transition in the protocol machine. BT requester and BT responder then enter a new protocol machine state that signifies that the business document has been exchanged. Once the BT responder has validated the business document and is ready to send a RA back, a new Web Services call (which corresponds to another protocol machine transition) is used to transmit the RA.

[10]`http://spinroot.com/`, last access: 12/20/2011

In summary, the design of the BT execution model is a mix of synchronous and asynchronous coordination in order to allow for simple protocol design on the one hand and decoupled processing of business content on the other hand. Only some calls to helper services may leverage the synchronous interaction style because its processing time can be assumed to be negligible.

At the BC level, transitions in the choreography model may correspond to several Web Services calls (see section 5.3). Similarly, BC level states such as BTAs and BCAs may represent the execution of several non-trivial interactions. Therefore, the distinction between synchronous and asynchronous communication as well as the distinction between synchronous and asynchronous interaction is inadequate for characterizing the execution concept of BCs. However, the basic paradigm that integration partners take transitions together in lock-step and agree upon the result of their interactions remains. In so far, denoting the execution semantics of BTs and BCs as *synchronous* is basically correct, but it refers to a higher-level concept than synchronous communication.

Availability of Advanced Security and Reliability Features For maintaining the synchronous communication paradigm as just described, the assumption of agreement upon the result of message transmissions at the messaging level must be validated. In addition, the availability of the security features that are assumed on the messaging level according to section 4.3 must be shown.

The *Secure WS-ReliableMessaging (SecRM)* scenario as defined in [44] provides the messaging-level implementation of the following BT configuration parameters (cf. section 4.3) in a mutual way: reliability (ebBP attribute *isGuaranteedDeliveryRequired*), confidentiality (*isConfidential*), integrity (*isTamperDetectable*) and authentication (*isAuthenticated*). The provision of the SecRM scenario's properties has formally been validated by two independent groups ([4] and [19]), but the practical availability cannot be taken for granted because it was not designed for the latest versions of WS-ReliableMessaging (WS-RM) [143], WS-Security (WS-Sec) [135], WS-Trust (WS-Trust) [146], WS-SecureConversation (WS-SecConv) [144], WS-ReliableMessaging Policy (WS-RM-Pol) [142] and WS-Security Policy (WS-Sec-Pol) [145] (although the features used do not deviate significantly). Moreover, the complexity of the protocol as well as the standards used is significant so that interoperability across WS stacks does not come easy. Even if interoperability is considered to be pivotal for Web Services according to [117, 122] this does not necessarily mean that implementations of WS-* standards indeed are implemented in an interoperable manner. Consistently, [106] stress that *"although one of the main purposes of the standard [i.e., a WS security standard] is to guarantee the interoperability between different platforms, it might be necessary to test it on the field."*

During the dissertation project of this work, the diploma thesis of Johannes Schwalb has been supervised who analyzed the implementability of the SecRM scenario and the interoperability of the corresponding WS-* standard implementations.

For analyzing the implementability of the SecRM scenario, he derived a current version of the corresponding WS-Policy assertions and tested the implementation for the GlassFish-openESB platform (as described above) and IBM's WebSphere platform. The detailed description of the scenario, the policy-based implementation as well as the test results are given in appendix C. While the relevant functionality is available for the GlassFish-openESB platform in a satisfactory manner, support on the IBM platform is insufficient. Even if using the proprietary features of the IBM platform instead of the WS-Policy based implementation (cf. appendix C) may reveal availability of the relevant functionality, the use of proprietary features in itself is unacceptable for a B2Bi setting.

Furthermore, the interoperable implementation of the underlying WS-Sec-Pol and WS-RM-Pol features between the GlassFish-openESB platform and the IBM WebSphere platform was tested. 169 test cases were analyzed ([183, 196]) and only 28 test cases proved to be implemented and fully interoperable. Even worse, it is not possible to exchange a SOAP message between the two platforms that is both, confidentiality and integrity protected. The GlassFish-openESB platform does not support XPath for identifying the SOAP message elements to be encrypted. Instead, it relies on using the `EncryptedParts` assertion of WS-Sec-Pol and assumes an `IncludeToken` value of `Never` for using X509 tokens (which is the only basic token type supported across both platforms). The WebSphere platform ignores any `IncludeToken` value and always inserts the token into the SOAP messages which is then rejected by the GlassFish-openESB platform. In addition, IBM WebSphere does not support the `TransportBinding` assertion of WS-Sec-Pol so that Transport Layer Security (TLS) encryption cannot be asserted either (see [183,196] for details).

Note that these interoperability issues are not simply due to having tested WS stacks that were released before the publication of the latest versions of WS-I's Basic Security Profile (BSP) [239] and Reliable Secure Profile (RSP) [240] in late 2010. Basically, those profiles do not help in avoiding the interoperability problems described in [183, 196] (see below). The profiles are complemented by test tools and sample applications, but the profiles are authoritative. In the standard document itself, the purpose of the BSP is described as follows:

> "*This document defines the WS-I Basic Security Profile 1.1, based on a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.*" [239]

Those clarifications and amendments become manifest in so-called *requirements* together with some explaining text and, in case of the RSP, test expressions for evaluating SOAP messages. In [6], one of the BSP editors explains that by using the requirements *"the BSP limits the set of common functionality that vendors must implement and thus enhances the chances for interoperability. This in return reduces the complexities for the testing of Web Services security."* Moreover, she explains that *"the security consideration statements provide guidance that is not strictly interoperability related but are testable best practices for security."*

For example, requirement *R2001* of the BSP says that *"A SENDER MUST NOT use SSL 2.0 as the underlying protocol for HTTP/S"*. The explanatory text justifies the requirement by pointing out that *"SSL 2.0 has known security issues and all current implementations of HTTP/S support more recent protocols"*.

A common characteristic of those requirements is that they define constraints on the level of SOAP messages, i.e., the existence, order and content of XML elements within SOAP messages or the actual exchange of SOAP messages is described. Consequently, the testing tools of the WS-I take SOAP messages as input and check them for compliance to the BSP and RSP requirements. From the perspective of facilitating interoperability, this amounts to replacing actual interoperability testing as described in [197] by checking standard compliance of SOAP messages. However, checking standard compliance itself is subject to errors and therefore merely an add-on to true interoperability testing but not a replacement. Even worse, the relation between WS-Sec-Pol/WS-RM-Pol assertions and the corresponding SOAP messages exchanged is not described in the BSP and RSP at all. In section 5.1.1, the BSP explicitly allows for out of band agreement for specifying the use of WS-Sec. Moreover, it states in several sections (9, 10, 13.1) that *"[..]no security policy description language or negotiation mechanism is in scope for the Basic Security Profile[..]"*. The RSP recommends (though not requires) the use of WS-RM-Pol for configuring the use of WS-RM in its section 2.4, but it does not define the relation between WS-RM-Pol assertions and SOAP messages either.
A detailed discussion in how far the BSP/RSP requirements could have helped in alleviating the detected interoperability issues between the GlassFish-openESB platform and the IBM WebSphere platform is available in [183], but, all in all, none of the core interoperability problems detected is due to violating WS-I profiles. Only 1 issue out of 10 core interoperability issues is reinforced by the BSP.

In summary, neither interoperable implementation of the relevant SecRM scenario functionality across different platforms nor a satisfactory implementation of the SecRM scenario on arbitrary platforms is available. Therefore, an alternative to using the GlassFish-openESB platform is needed. For the security part, tunneling HTTP via TLS is an option as TLS is a commonly available protocol and the properties of TLS have been validated in various scientific publications. Among others, Paulson [156] was the first to formally validate TLS in terms of authentication, privacy and message integrity. However, the reliability features of Transmission Control Protocol (TCP) which is typically the basis of TLS implementations cannot be taken for granted because TCP messages are at lower protocol level than TLS and an attacker must be assumed to try to attack lower protocol levels as well. Of course, arbitrary security goals cannot be broken like that, but the reliability feature of TCP is at risk. Paulson consistently states in his analysis of the matter that *"once session keys are established, the parties have a secure channel upon which they must run a reliable communication protocol"* [156]. For this purpose, running bare WS-RM on top of TLS is an option as WS-RM is significantly less complex than the SecRM scenario. In consequence, the availability of the necessary security and reliability features for maintaining the synchronous communication semantics as postulated above can be taken for granted, even if it does not come cheap.

5.2. Implementation of BusinessTransactions

This section describes how the BT execution model of section 4.3 together with the defined execution semantics that ensures consistent outcome between the requester and responder role can be implemented. Of course, this is not the only valid way of implementing the model, but it demonstrates the feasibility of implementing the BT execution model using BPEL-based control processes.

As example, the implementation of a One-Action-BT that exchanges a RosettaNet *Pip3A20PurchaseOrderConfirmationNotification* business document and makes use of a RA, AA, and the retryCount BT configuration parameter is shown. As much as the execution model introduced in section 4.3 can be extended for alternative BT configurations, the implementation proposed here can be extended as well. The description below will show how to explicitly represent the states of the BT protocol machines in BPEL so that deriving implementations for different BT configurations essentially is just a matter of adding/removing state representations and adjusting some variables.

For implementation, additional design choices beyond those of the last section and section 4.3 have to be made. Firstly, the message header defined in listing 5.1 may be applied to local message exchanges only, i.e., between the backend, master and BT control process of one integration partner, or additionally to the message exchanges between the requester and the responder control process. Not using the message header for cross control process messages necessitates the availability of message fields for mapping correlation information within those messages. For the RosettaNet business documents like Pip3A20PurchaseOrderConfirmationNotification as well as for business signals and business signal exceptions as defined in the ebBP standard, such fields are available. The advantage of not using the message header for cross control process communication is that existing implementations can be migrated without breaking existing partner interfaces. However, the correct usage of correlation information must be ensured. As *not* applying the message header to cross control process communication is the more challenging strategy, the implementation proposal below only applies the message header to local exchanges. Therefore, the message types of section 4.3's execution model may be transmitted in either *wrapped* form, i.e., with message header, or in *raw* form.

The second design choice to be taken is whether superordinate master control processes should be assumed for the BT level control processes. As a single BT execution does not make up a complete B2Bi choreography, such master control processes are assumed. If the B2Bi choreography only consists of one single BT execution, then either the backend can take over the master role or the simplified execution models developed for the MCC phase 1 Web Services profile [172] can be adopted.

Before the structure of the BPEL control process implementations is described, the most important concepts for mapping the BT execution model elements are

Msg. Type	Usage
TxStart	Used by master control processes to start the requester and responder control processes.
SolBizDoc	Used by the requester to solicit the business document from the backend.
BizDoc	Carries the business document and is transmitted by the requester backend to the requester control process and by the responder control process to the responder backend in wrapped form. Sent in raw form from the requester control process to the responder control process.
RA	Carries the ReceiptAcknowledgement and is transmitted by the RAC service to the responder control process and from the requester control process to the requester backend in wrapped form. Sent in raw form from the responder control process to the requester control process.
RAE	Carries the ReceiptAcknowledgementException and is transmitted by the RAC service to the responder control process and from the requester control process to the requester backend in wrapped form. Sent in raw form from the responder control process to the requester control process.
AA	Carries the AcceptanceAcknowledgement and is transmitted by the responder backend to the responder control process and from the requester control process to the requester backend in wrapped form. Sent in raw form from the responder control process to the requester control process.
AAE	Carries the AcceptanceAcknowledgementException and is transmitted by the responder backend to the responder control process and from the requester control process to the requester backend in wrapped form. Sent in raw form from the responder control process to the requester control process.
GE	Carries the GeneralException and is transmitted between the requester control and responder control process in raw form in both directions. Transmitted by the requester control process to the requester backend and by the responder control process to the responder backend in wrapped form.
Cancel	Transmitted by the requester backend to the requester control process and from the responder backend to the responder control process for canceling the BT execution.
TxResult	Used by control processes to report the result of a BTA to the master processes. The result is a generic protocol result as defined by ebBP's ConditionGuardValue language or, in case of ProtocolSuccess, the generic protocol result and the exchanged business document.

Table 5.1.: BT Protocol Messages Overview

introduced. The states of the BT protocol machines of section 4.3 define what kind of events are admissible. For each of the states, either a single message receive event, a single message send event, concurrent receive events, or one send event and one or more concurrent receive events can be admissible. An additional variant for each of these cases may be defined by adding one or more concurrent timeouts.

Send events are implemented using BPEL's `invoke` construct and single receive events by means of BPEL `receives`. In case of concurrent receive events and timeouts, BPEL's `pick` construct is used to discriminate between the concurrent events where `onMessage` branches are used for incoming messages and `onAlarm` branches for timeouts. Concurrent send and receive events as well as timeouts are implemented by enabling the send event on the one hand and the receive events and timeouts on the other in an alternating manner (see below). BPEL `partnerLinks` are used for identifying communication partners based on WSDL interfaces and for each communication partner defined in the protocol machines two WSDL files (cf. last section) and two `partnerLinks` are defined in order to cover each communication direction separately. Except of the *wrapped/raw* distinction introduced above, the message types used for the BPEL implementation correspond to the message types of section 4.3's execution model. Thus, instead of discussing all defined WSDL interfaces, table 5.1 summarizes the types of messages used as well as which format is used for which communication relationship. Note that message types that are only exchanged locally, always use the message header of listing 5.1. Messaging failure events are captured using BPEL `faultHandlers` and `catch` elements. Beyond that, control flow definition is based predominately on BPEL's `sequence`, `if` and `while` constructs. All in all, a subset of common BPEL constructs is used the availability of which can be taken for granted on virtually any BPEL engine.

Listing 5.4 shows the rough BPEL structure that implements the requester control process. For discussing this BPEL process under consideration, the term *focal BPEL process* will be used. First, some structural definitions are made beginning with several `import` tags for importing the WSDL interfaces that define the communication options with the backend, master control process and responder control process as well as the correlation set properties. Based on the `partnerLinkType` definitions in the WSDL files, the `partnerLinks` that the focal BPEL process uses for communication are defined. For example, the `partnerLink` *'FromMaster'* in line 4 defines, by using the `myRole` attribute, that the focal BPEL process takes the *'PIP3A20RequestorMasterClientPortTypeRole'* and is able to consume the messages defined in the associated WSDL `portType` (the `partnerRole` tag would imply the opposite communication direction). Subsequently, several variables for storing messages and controlling control flow are defined and *'corrSetTransaction'* is defined as the only correlation set of the BPEL process based on the *'prop_CollaborationIdentifier'* property of listing 5.2. Then, the BPEL `faultHandlers` construct is used to define global BPEL `catch` elements for installing different error handling routines that cover the various types of errors defined in the requester protocol machine. In line 14 a user-defined *'deliveryException'* is caught that is thrown when the business document could not be delivered several times. Whenever an error occurs throughout the process definition and is not handled by lower-level `faultHandlers`, then it is propagated to the `faultHandlers` definition of the root `process`. By throwing user-defined exceptions the logic contained in the respective `catch` blocks can be used whenever needed. This is used as a convenient way for implementing states *'RES-GE'*, *'CP-GE'*, *'CP-GE2'*, *'BE-GE'*, *'DelRAE'* and *'DelAAE'* of the requester's protocol machine (cf. figure 4.2). These states essentially capture the various failure paths of the requester's control process that arise from TTP, TTRA and TTAA timeouts (cf. BT execution model), from repeated business document delivery errors, from exception messages sent by the responder (RAE, AAE, GE) and cancel messages sent by the backend.

Thereafter, the root sequence *'mainSeq'* follows that covers the actually intended path of a BT execution. First, the *'TxStart'* message of the master control process that initializes the *'corrSetTransaction'* correlation set is awaited in line 20. Then several variables are initialized (omitted in listing 5.4) and the *'DeadlineCreator'* helper service is used to calculate the deadline for the TimeToPerform (TTP) value of the current process instance, i.e., the TTP value is simply added to the current time by the helper service. This deadline then will be used in several deadline-based `onAlarm` constructs. Note that this does not cause any problems with respect to the execution semantics as typical TTP values range from several minutes to hours and the calculation of such a deadline is a matter of milliseconds. The following BPEL `while` construct named *'processLoop'* starting in line 28 is used to switch across the states of the requester protocol machine that make up the intended flow of interaction, i.e., *'Started'*, *'AwaitBizDoc'*, *'DeliverBizDoc'*, *'AwaitRA'*, *'DelRA'*, *'AwaitAA'*, *'DelAA'* and *'Propagate'*. A simple string variable *'procState'* is used to represent the current state of the requester's protocol machine and a series of BPEL `if` constructs is used to check for the admissible states. If the condition for a particular state is true, e.g., if the requester is in state *'Started'* as checked in line 32, then the implementation of the respective state is included within the `if` structure (see listings 5.5, 5.6 and 5.7 for examples).

Note that, for the purpose of conciseness, some details of XML tags in listing 5.4 and the following listings are hidden. So, attributes such as namespace declarations or the several attributes for determining a WSDL operation that corresponds to a receive/send event frequently are left out. Moreover, incomplete tags are sometimes represented as <tagname> to show that one or more lower level tags of `tagname` are missing and <tagname>* in case multiple occurrences of `tagname` are to be inserted. Obviously, these conventions violate the XML standard. Moreover, please note that the `name` attribute included in several BPEL tags is irrelevant for the control flow of the process and has been added for the sole purpose of enhancing readability.

Listing 5.4: Rough Structure of the Requester Process

```
1  <process name="composablePIP3A20-Requestor">
2   <import>* <!-- Import various WSDL files for interface/correlation definition -->
3    <partnerLinks>
4      <partnerLink name="FromMaster" myRole="
           PIP3A20RequestorMasterClientPortTypeRole" ..> <!-- One partnerLink per
           peer service and communication direction -->
5      <partnerLink>*
6    </partnerLinks>
7    <variables>
8     <variable>* <!-- Variables for storing messages and BT configuration parameters
           are defined at process scope -->
9    </variables>
10   <correlationSets>
11       <correlationSet name="corrSetTransaction" properties="
             ns0:prop_CollaborationIdentifier"/>
12   </correlationSets>
13   <faultHandlers>
14     <catch faultName="procFaults:deliveryException">
15      ...
16     </catch>
```

```
    <catch>* <!-- Fault handlers for the other five exception types like ttarErr, ttaaErr etc. -->
  </faultHandlers>
 <sequence name="mainSeq">
    <receive createInstance="yes" partnerLink="FromMaster" operation="txStartOp">
       <correlations>
          <correlation set="corrSetTransaction" initiate="yes"/>
       </correlations>
    </receive>
    ... <!-- Initialize some variables -->
    <invoke partnerLink="DeadlineCreator" operation="createDeadline">
    <assign name="setTTP"> <!-- set the global variable for the TTP deadline -->
  <while name="processLoop">
   <condition>$processNotDone</condition>
   <sequence name="processSeq">
    <if name="If-Started">
     <condition>$procState = 'Started'</condition>
      ... <!-- Implementation of Started state -->
    </if>
    <if name="If-AwaitBizDoc">
     <condition>$procState = 'AwaitBizDoc'</condition>
      ...
    </if>
    ... <!-- Implementations of the other protocol machine states -->
   </sequence>
  </while>
 </sequence>
</process>
```

Listings 5.5, 5.6 and 5.7 exemplify the most important types of combining send, receive and timeout events and therefore give a good overview of how states of a BT protocol machine are to be implemented. The trivial cases of an isolated send or an isolated receive event are left out.

Listing 5.5 shows the implementation of the requester's started state where sending the *'SolBizDoc'* message to the backend, receiving the *'GE'* message from the responder and the *TTP* timeout are admissible. Apparently, the implementation of the *'Started'* state only consists of an `invoke` for sending the *'SolBizDoc'* message to the backend (line 4) and an assign for switching to the next state in the next iteration of the *'processLoop'* (line 5). That means, that receiving the *'GE'* message and firing the *'TTP'* timeout is disabled as long as the *'SolBizDoc'* message is tried to be delivered.

Note that this is well acceptable for the BT execution semantics. Most importantly, the consistent outcome of a BT execution across integration partners is not threatened for two reasons. Firstly, message transmission failures explicitly are part of the model and thus disabling receive events temporarily does not do any harm. Secondly, timeouts are controlled by one partner and then communicated to the corresponding control process.

Beyond not threatening the consistency property of the BT execution model, temporarily disabling receive and timeout events also is not harmful from a practical perspective. Firstly, the transmission of a message typically does not take a lot of time. So, the message transmission of a communication partner typically will not time out before the message of the focal partner is transmitted. If the message transmission of the communication partner is not admissible in the next state any more, then it is as if the message send event of the focal process occurred beforehand which is valid. Secondly, if a timeout occurs before the message transmission of the focal process is completed then it will immediately be fired once the corresponding `onAlarm` of the next `pick` is activated (cf. [137, section 11.5]). If the timeout is not admissible in the next state any more, then it is as if the message send event of the focal process occurred beforehand which is valid again. Note, in that regard, that firing a timeout some seconds sooner or later does not do any harm because of the typical time horizon of BT timeouts.

Finally, note that BPEL's `eventHandlers` construct deliberately is not used, although it seems to be an obvious solution for dealing with concurrent events. The reason is that an event handler is executed in parallel to the `scope` (cf. [137, section 12.7]) it is attached to (parent scope in the following). Let's assume that some event X is enabled in some scope and that some event Y is enabled by means of a message event handler attached to the scope. If Y fires and the child scope of the event handler starts then X is still enabled (even if this is for a very short time because a `throw` construct is used in the event handler to interrupt its parent scope). If X then is fired the BT execution semantics is broken because events are defined to mutually exclude each other. For example, assume that the requester control process is in state *'AwaitAA'* and that X corresponds to the receipt of the responder's *'AA'* message. Furthermore, assume that Y corresponds to the receipt of the backend's *Cancel* message. If both messages are accepted and backend and responder control process terminate in a very short time (which would be admissible according to the BT execution model) then the interacting parties would end up in inconsistent states.

Listing 5.5: Implementation of Requester's 'Started' State

```
<if name="If-Started">
 <condition>$procState = 'Started'</condition>
 <sequence name="Sequence-Started">
  <invoke partnerLink="ToBackend" operation="txSolicitBDOp">
  <assign name="Assign-Started">
   <copy>
    <from>'AwaitBizDoc'</from>
    <to variable="procState"/>
   </copy>
  </assign>
 </sequence>
</if>
```

Listing 5.6 shows the implementation of the requester's *'DeliverBizDoc'* state where, again, a send event competes with receive events and timeout events. The difference to listing 5.5 is that the protocol message (*'BizDoc'*) is sent to the responder control process instead of the backend. For cross control process communication, the retry count of the BT configuration must be considered and therefore the `invoke` construct for sending the business document (line 42) is included in a `while` loop (line 4). In case of a communication error, the openESB BPEL engine raises a *'systemFault'* that is caught in line 8 in a BPEL `faultHandlers/catch` declaration. Then the *'errCount'* variable that represents the number of retries is incremented (line 10) and checked against the maximum number of retries (line 12). If the maximum number of retries has been exceeded a user-defined exception is thrown (line 13) which causes the *'retry'* `while` loop to be left and is eventually caught by one of the root level fault handlers of listing 5.4. Otherwise, the next send try is not launched immediately, but a backoff strategy is applied to circumvent temporary unavailability of the responder control process. During this time the other competing events of state *'DeliverBizDoc'* are enabled and a BPEL `pick` construct is used to select exactly one of these (cf. [137, section 11.5]). Thus, standard BPEL functionality is used to comply with the execution semantics for BTs that only allows one event to be processed for a particular state. Two `onMessage` constructs and one `onAlarm` construct are used to allow for a *'GE'* of the responder, a *'Cancel'* of the backend and a *'TTP'* timeout. In each case, a `throw` construct (lines 20, 25 and 29) is used to raise a user defined exception that then is caught and processed by the root process' fault handlers. The last `onAlarm` in line 31 implements the back-off strategy of waiting for 60 seconds before trying the next delivery of the *'BizDoc'* message. When this `onAlarm` is fired nothing particular is done and thus just the default semantics of a fault handler is applied which is terminating the parent `scope` (the *'bizDocDeliveryScope'* `scope` of line 6 in this case). Then a new iteration of the *'retry'* `while` loop is started. The alternative to leaving this `while` loop by means of an error is successfully delivering the *'BizDoc'* message. If so, the *'bizDocDelivery'* variable for controlling the *'retry'* loop is set correspondingly, the deadline for *'TTAR'* and *'TTAA'* are calculated and the next state is switched to (line 45 onward).

Listing 5.6: Implementation of Requester's 'DeliverBizDoc' State

```
1  <if name="If-DeliverBizDoc">
2   <condition>$procState = 'DeliverBizDoc'</condition>
3   <sequence name="Sequence-DeliverBizDoc">
4    <while name="retry">
5     <condition>not($bizDocDelivery)</condition>
6     <scope name="bizDocDeliveryScope">
7      <faultHandlers>
8       <catch faultName="sxeh:systemFault">
9        <sequence name="errCountSeq">
10        <assign name="incErrCount"> <!-- increment errCount variable -->
11        <if name="If1">
12         <condition>$errCount &gt;= $maxRetries</condition>
13         <throw name="throwDelError" faultName="procFaults:deliveryException"/>
14         <else>
15          <pick name="Pick-DeliverBizDoc">
16           <onMessage partnerLink="FromResponder" operation="ExceptionOp">
17           <correlations> <!-- use corrSetTransaction correlation set -->
18           <sequence name="DealWithPartnerErrSeq">
19            <assign name="copyErrSrc"> <!-- Copy error details -->
20            <throw name="throwPartnerEx" faultName="procFaults:partnerException"/>
21           </sequence>
22           </onMessage>
23           <onMessage partnerLink="Backend2Self" operation="txCancelOp">
24            <correlations> <!-- use corrSetTransaction correlation set -->
25            <throw name="throwCancel" faultName="procFaults:cancelException"/>
26           </onMessage>
27           <onAlarm>
28            <until>$deadlineTTP</until>
29            <throw name="throwTTP" faultName="procFaults:ttpException"/>
30           </onAlarm>
31           <onAlarm>
32            <for>'P0Y0M0DT0H0M60.0S'</for>
33            <empty name="backOff"/>
34           </onAlarm>
35          </pick>
36         </else>
37        </if>
38       </sequence>
39      </catch>
40     </faultHandlers>
41     <sequence name="deliverySeq">
```

```
      <invoke partnerLink="ToResponder" operation="
          Pip3A20PurchaseOrderConfirmationNotificationOp">
       <correlations> <!-- use corrSetTransaction correlation set -->
      </invoke>
      <assign name="setCommSuccess"> <!-- No exception means success -->
       <copy>
        <from>true()</from>
        <to variable="bizDocDelivery"/>
       </copy>
      </assign>
      ... <!-- Compute deadlines for ttAA and ttAR -->
      <assign name="Assign-DeliverBizDoc"> <!-- Switch to AwaitRA state -->
     </sequence>
    </scope>
   </while>
  </sequence>
</if>
```

Listing 5.7 shows the implementation of the requester's *'AwaitRA'* state where several receive events compete with several timeout events. From a conceptual point of view, there is no difference to the `pick` construct of listing 5.6 and listing 5.7. Note however, that the same variable *'deadlineTTP'* for controlling the *'TTP'* timeout is used. This is possible by calculating the deadline once the corresponding timer is activated (cf. listing 5.4) and using an `onAlarm/until` combination instead of an `onAlarm/for` combination.

Listing 5.7: Implementation of Requester's 'AwaitRA' State

```
<if name="If-AwaitRA">
 <condition>$procState = 'AwaitRA'</condition>
 <sequence name="Sequence-AwaitRA">
  <pick name="Pick-AwaitRA">
   <onMessage partnerLink="FromResponder" operation="ReceiptAcknowledgementOp">
    <correlations> <!-- Use corrSetTransaction correlation set -->
    <sequence>
     <assign name="prepReceiptAck"> <!-- Store RA -->
     <assign name="Assign-AwaitRA"> <!-- Switch to DelRA state -->
    </sequence>
   </onMessage>
   <onMessage partnerLink="FromResponder" operation="ExceptionOp">
     <!-- Deal with PartnerError -->
   <onMessage partnerLink="Backend2Self" operation="txCancelOp">
     <!-- Deal with backend's cancel -->
   <onAlarm>
    <until>$deadlineTTP</until>
    <throw name="throwTTP-AwaitRA" faultName="procFaults:ttpException"/>
   </onAlarm>
   <onAlarm>
    <until>$deadlineTTAR</until>
    <throw name="throwTTAR" faultName="procFaults:ttarException"/>
   </onAlarm>
   <onAlarm>
    <until>$deadlineTTAA</until>
    <throw name="throwTTAA" faultName="procFaults:ttaaException"/>
   </onAlarm>
  </pick>
 </sequence>
</if>
```

Note that the description of implementing the sample requester control process for the PIP3A20-based BT easily can be adapted to other BT configurations. Moreover the concept for implementing a responder control process is strikingly similar as the underlying paradigm for specification, a communicating state machine, is exactly the same.

Testing the requester and responder control processes on the GlassFish-openESB platform is relatively easy because only a common and non-intricate subset of BPEL constructs is used for implementation. For example, control `links` between the activities of a `flow` construct or a parallel `forEach` construct that are not supported by all contemporary BPEL engines are not used. Backend and Master callback interfaces have been implemented using EJBs and soapUI mock servics as convenient. The only problem that was detected was a bug in the openESB binding between its BPEL engine and HTTP component. In case a receive event is enabled in two subsequently activated `picks` or, more generally, incoming messaging activities, then the second receive event may not get activated on a nondeterministic basis. While this increases the testing burden to some extent, it does not affect the validity of the proposed implementation. Additional tests revealed that this behavior is not existent on Oracle's BPEL engine. Note that the state machine model of BT control processes on the one hand and explicitly representing states as `if` constructs in a `while` loop greatly simplifies testing because the reachability of each state and the possibility of firing each enabled transition then can easily be derived.

Apart from the practical validation of the BT execution model, a formal analysis of the BT execution protocol has been done using the SPIN model checker. The protocol machine definitions in Promela (SPIN's input language) as well as the consistency property formalization and analysis have been performed by Matthias Geiger during his diploma project that has been supervised in the course of this work's dissertation project. Therefore, the results of the SPIN validation are available in appendix D.

5.3. Implementation of BusinessCollaborations

This section demonstrates how the execution semantics of ebBP-Reg can be implemented using BPEL. As discussed in the introduction of this chapter, ebBP-Reg is the more demanding model compared to the other binary choreography style ebBP-ST (SeqMP is not supposed to serve as model for generating implementations). However note that the underlying paradigm of both, ebBP-Reg and ebBP-ST, is a state machine so that adapting the implementation guidelines of this section to ebBP-ST is not very hard. Moreover, a prototypic translator of ebBP-ST models has been presented in [160, 182] and the corresponding mapping rules are given in appendix B.

Differences between Implementing ebBP-Reg Models and BTs The implementation of BTs is similar to the implementation of BCs in the sense of implementing a state machine-based model. However, there are fundamental differences with respect to the variability of models, implementation details and content to be exchanged.

The set of ebBP-Reg models is unbounded whereas the number of ebBP BT configuration instantiations is limited (except for timeout and retry values). Therefore, mapping rules for the individual ebBP-Reg constructs are needed instead of an implementation template that can be adapted to the individual BT configuration instantiations. As a consequence, using a fixed set of global variables is inadequate for mapping ebBP-Reg models and BPEL `scopes` are used for defining locally visible variables per ebBP-Reg construct.

Implementing process correlation also is slightly different from the BT strategy. Firstly, the message header of listing 5.1 is applied to all collaboration level message exchanges including cross-organizational message exchanges (whereas business documents and signals may be exchanged in *raw* form for BTs). This is valid because ebBP-Reg is a new model that requires a dedicated set of message types that have to be exchanged in a certain order. As existing collaboration level message exchanges are likely to be built upon a different set of message types and exchange orders, including the message header of listing 5.1 in the adaptation of existing collaboration level implementations is acceptable. Secondly, the hierarchical organization of ebBP-Reg models affects binding of correlation properties. For BT execution, the assumption is that backend, master and partner control processes all use the same instantiation of the message header for communicating with a particular control process. Incoming messages then all are mapped to the intended BPEL process by mapping the *'InstanceIdentifier'* field of the message header. For ebBP-Reg, messages from a lower-level process must be considered that use the message header instantiation of the lower nesting level. Hence, an incoming message of a lower-level process must be mapped using the *'ParentIdentifier'* field of the message header as shown in listing 5.8 instead of the *'InstanceIdentifier'* field as shown in listing 5.3.

Listing 5.8: Sample Correlation Property Alias for Binding of Incoming Messages from a Lower Level Control Process

```
<vprop:propertyAlias
 propertyName="corr:prop_CollaborationIdentifier"
 messageType="tns:bc-single-3A19CbResultMsg"
 part="bc-single-3A19CbResultPart">
    <vprop:query>mb:CollaborationMetaBlock/mb:ParentIdentifier</vprop:query>
</vprop:propertyAlias>
```

The last major difference in implementing ebBP-Reg models compared to BTs is the type of content to be exchanged. Collaboration level messages do not carry business content. At best, the need for exchanging business relevant content is communicated via collaboration level messages, but all relevant business documents and business signals that carry meaning from a business point of view are exchanged within BTs. The purpose of collaboration level messages is exchanging technical details like correlation information and coordinating upon collaboration control flow in order to ensure strict compliance to the respective ebBP-Reg model. This is done by means of applying so-called *"micro-protocols"* that are specific for each ebBP-Reg control flow construct and range in complexity from basically trivial for sequential structures and complex for event-based choices. Note that those micro-protocols do not trigger activities on their own, but just serve as technical arbitrators for backend triggers. The backends actually request the execution of particular activities and the micro-protocols implement these across all related parties in a consistent manner. In case of concurrent requests, micro-protocols ensure a global order of trigger events so that diverging control flow is avoided. In order to do so, one out of each pair of BC control processes for implementing a BC is assigned the *"leader"* role whereas its peer is assigned the *"non-leader"* role (see below for examples). This is a merely technical decision and, as there is no global time in distributed systems anyway, does not affect validity.

As collaboration level messages do not carry business information, the assumptions about the communication channel can be rethought as well. From a reliability point of view, loosing such a message is not really serious because the worst thing that can happen is a stalled process. However, if an attacker is part of the failure model, the application of the SecRM scenario or similar means as discussed in section 5.1 is advisable.

Overview of Implementing ebBP-Reg Models The implementation of ebBP-Reg models follows the pairwise control process modularization scheme described in section 5.1, in particular figure 5.1, and the WSDL interfaces are designed correspondingly (one WSDL definition for each direction of a communication link). Instead of discussing all those WSDL interfaces, table 5.2 summarizes the collaboration level message types exchanged.

The basic interaction style for exchanging these messages is asynchronous except for three synchronous interactions that are used by BC control processes for requesting new ids for lower level control processes (messages *'idReq'*, *'idRes'*), for requesting TimeToPerform values (messages *'ttpReq'*, *'ttpRes'*), and for evaluating expressions against BT results (messages *'evalReq'*, *'evalRes'*). Note that these interactions do not require human interaction and are performed within the safe local environment of one partner. Hence, they can be assumed to take very little time.

Conversely, the communication style again is synchronous for simplifying micro-protocol design.

The derivation of BPEL control processes from ebBP-Reg choreographies is described below by first discussing how the overall process structure looks like and then describing how the various ebBP-Reg structures are translated into this structure.

The various listings are taken from the implementation of the use case depicted in figure 4.11 of section 4.5. This use case covers all major control flow constructs of ebBP-Reg and therefore also provides reasonable validation of the mapping rules described. For conciseness, the same violations of XML syntax used in the last section are reapplied again.

For setting up a BC control process, similar static configurations in terms of WSDL `imports`, BPEL `partnerLink` definitions and correlation sets are needed as for BT processes. A more detailed description therefore is omitted. For starting the behavioral definition of a BC control process, the implementing BPEL process uses a BPEL `receive` to await a `cbStart` message from either a parent control process

Ctrl. Msg.	Explanation
cbStart	Used by a BC control process to start a child control process.
initFail	Used by a child BC control process to signal to a parent control process that the peer BC control process is not reachable.
reqAct	Used by either a backend or a partner BC control process to request the execution of an activity at the *leader* control process.
actChoice	Sent by a BC control process to signal that a new lower level activity shall be started. If both BC control processes may start a lower level activity, only the leader process is allowed to send this message. If concurrent requests of backends are possible, the control processes use this message to inform the backends about the chosen activity.
initAct	Used by a BC control process to distribute the instance id of a new activity to its peer control process.
initAck	Used by a BC control process to acknowledge the start of an activity to the partner control process.
idReq	Used by a BC control process to request a new activity id at the backend.
idRes	The synchronous reply message to an idReq message.
ttpReq	Used by a BC control process to request the ebBP timeToPerform value for a new activity (if not statically assigned) at the backend.
ttpRes	The synchronous reply message to a ttpReq message.
evalReq	Used by a BC control process to request the evaluation of a given expression against the result document of a BT execution at the backend.
evalRes	The synchronous reply message to a evalRes message.
actReady	Used by a BC control process to inform its backend that a BTA will be set up immediately or that a child BCA has been set up. The distinction with respect to sending the actReady message before or after setup is aligned with the first message to be exchanged for the respective activity. In BT executions, the first message between control processes and backends is sent by the control processes. Conversely, the first message exchanged between backends and control processes for a BC execution is sent by the backend.
bcResult	Upon termination, used by a BC control process to report the BC result to its backend and its parent control process (if existent).
txResult	Used by a BT control process to report the BT result to its parent control process.
txStart	Used by a BC control process to start a child BT control process.

Table 5.2.: BC Control Messages Overview

(if existent) or from the backend. In the latter case (which means the process under consideration is a root level process), the `cbStart` message is forwarded to the corresponding partner control process which in turn informs its backend component about the new collaboration instance. Upon receipt of the `cbStart` message, the correlation sets for identifying the process instance are set up (cf. section 5.1). In case the partner control process is not reachable, an `initFail` message is returned to the initiator and the BC control process is terminated. If the collaboration level ebBP `timeToPerform` value has been set to `runtime` the BC control process requests a `ttpRes` message from its backend using a `ttpReq` message and propagates it to its peer.

Then, the main BPEL *scope* of the process is entered that carries a BPEL `onAlarm` event handler for controlling the collaboration's `timeToPerform` timer. If the timeout occurs, the event is stored in a boolean variable (*'interrupted'* in listing 5.9) and once the next activity completes, the control process terminates and sends a corresponding `bcResult` message to its initiator informing it about the technical failure (see listing 5.16). The content of the main BPEL `scope` consists of a BPEL `while` that continues until a final state is reached. Within this loop, a series of BPEL `if`s are used to determine the *state* the collaboration currently is in (variable *'cbState'*). Any BCA, BTA, parallel structure, event-based choice and final state that is not located within a parallel structure is considered to be a state (cf. section 4.5). Looking at the use case depicted in figure 4.11, the parallel structure at the top or the BTA `BT-3A21` would be considered as states, but not BCAs `BC-single3A19-1/2`. In each of these states, i.e., within the BPEL `if` tags, the code for the respective ebBP-Reg language constructs is mapped. Listing 5.9 exemplifies the concept.

Listing 5.9: Main loop of a BCA control process

```
<while name="cbWhile">
  <condition>not($inEndState) and not($interrupted)</condition>
  <sequence name="cbSwitchSeq">
    <if name="cbSwitch-fk-parallel">
      <condition>$cbState = 'fk-parallel'</condition>
      <scope name="scope-fk-parallel">
       ... code implementing the parallel structure ...
      </scope>
    </if>
    <if name="cbSwitch-bta-bt-PIP3A20">
      <condition>$cbState = 'bta-bt-PIP3A20'</condition>
      <scope name="scope-bta-bt-PIP3A20">
       ... code implementing PIP 3A20 ...
      </scope>
    </if>
    ... switch across other 'states' ...
  </sequence>
</while>
```

Eventually, when a final state has been reached (businessSuccess, techFail or businessFail for the above use case) or an interrupt has occurred then the global loop condition is set to false, the parent control process and/or the backend process are informed about the result and the process is terminated.

Mapping Rules for ebBP-Reg Models The two BPEL control processes that are derived for each nesting level of an ebBP-Reg model by applying the below mapping rules are not identical. Some mapping rules depend on whether the BPEL process to be generated takes the *leader* role or the *non-leader* role as discussed above. This implies that the leader role is fixed at build-time, but the rules could easily be extended such that a particular process implements both roles and then a leader election algorithm could be used at run-time. Remind further that decisions and loops in an ebBP-Reg model are implicitly covered by the evaluation of the outgoing transition guards of BTAs and BCAs. Hence, there are no mapping rules for decisions and loops. Also note that, by leveraging standard component interfaces as described in section 5.1, the resulting BPEL processes are fully executable.

Mapping Rule 5.3.1 (Start)

Essentially covered in the description of the process setup above. This rule does not depend on the *leader/non-leader* distinction.

Mapping Rule 5.3.2 (BTA)

This rule is applied if only one single BTA is eligible for execution and does not depend on the *leader/non-leader* distinction. However, the BPEL mapping depends on whether the focal control process takes the *requester* or the *responder* role of the BT. Listings 5.10 and 5.11 show the BPEL mapping for the former and the latter case, respectively. Remember that the BPEL code described has to be inserted into one of the BPEL `if` branches of the main process `while` loop as described in listing 5.9. Consistently, both listings start out with the definition of a BPEL `scope` and the definition of several local variables that are needed for the implementation of the micro-protocol that ensures control flow (trivial for this case).

If the control process maps to the requester role of the BT (listing 5.10), then the first messaging activity corresponds to waiting for the backend to request the execution of the respective BTA (BT-3A20 in this case) using a *reqAct* message (see line 10 of listing 5.10). Upon receipt of the *reqAct* message, the control process synchronously queries a new activity id for the BTA (line 16), informs the partner control process that the execution of the BTA has been requested using an *actChoice* message (line 18), and then distributes the BTA's activity id to the partner control process (line 20). Then the *initAck* message of the partner is waited for that is only sent if the partner has set up the BTA's responder control process. This sequence is not harmful because the responder control process waits for the requester control process of a BT to send the first cross-organizational message at the BT level. Moreover, introducing the exchange of the *initAck* message enables BC level routing even for BT protocol failures because the BT responder protocol process will eventually determine a protocol failure in case that the requester control process does not send a message. As the BT execution protocol is known to always produce a consistent outcome (cf. appendix D), the focal BC control process can take routing operations as defined in the ebBP-Reg model even in case of errors. Note that this is different from other approaches that do not have separate initiation and execution phases because an execution phase error then may not be detected by both interacting parties. For example, the responder may not even notice that the requester tried to perform a BT in these alternative approaches.

Therefore, once the focal control process has received the *initAck* message of its partner, the backend is informed about the setup of the BT level control process. Subsequently, the BT level requester control process is started. This order of first informing the backend and then setting up the requester control process is sensible because, in the good case, the requester control process first solicits the business document from the backend and then receives the corresponding message from the backend. Note that starting the BT requester control process is a local action and therefore is unlikely to fail. Upon termination, the requester control process sends back a *txResult* message to the parent control process (line 31 of listing 5.10). This result then is evaluated in lines 33 to 71 where the generic protocol outcome according to ebBP ConditionGuardValue semantics is evaluated first. In case any kind of protocol failure is detected (lines 34 to 36) then the *'cbState'* variable is set to *'techFail'* which is the label of an end state in figure 4.11. The corresponding scope of that end state then is reached in the next iteration of listing 5.9's `while` loop. If no protocol failure is detected a helper service is used to evaluate the content-based expressions of the ebBP-Reg model against the business document exchanged during the BT execution (the business document is part of the result variable). As BPEL engines are required to support the XPath 1 standard, a separate helper service is actually not always necessary. However, for better debugging and better flexibility in choosing alternative expression languages, a dedicated service is advisable. The assumption is that the evaluation of expressions results in a boolean value and the next state of the ebBP-Reg model is switched to if the respective expression evaluates to true. If no expression evaluates to true, a corresponding exception is thrown (line 62). Note that ebBP-Reg requires the set of condition guards attached to the outgoing transitions of a BTA to be complete and disjoint. Therefore, the nested structure of BPEL `ifs` for evaluating content-based expressions could be replaced by a sequence of `ifs` as well. However, this would unnecessarily waste computing time.

Listing 5.10: Initiating a BTA if Collaboration Role Maps to BTA Requester

```
1  <scope name="scope-bta-bt-PIP3A20">
2   <variables> <!-- provide local variables for implementation of micro protocol
        message exchanges -->
3    <variable name="initAck-bta-bt-PIP3A20-FromPartner"
4      messageType="tns:cbInitAckMsg"/>
5    <variable name="reqAct-bta-bt-PIP3A20-FromBackend"
6      messageType="tns:cbRequestActivityMsg"/>
7    <variable>*
8   </variables>
9   <sequence name="seq-bta-bt-PIP3A20">
10   <receive partnerLink="FromBackend" operation="cbRequestActivityOp">
11    <correlations>
12     <correlation set="corrCollaboration" initiate="no"/>
13    </correlations>
14   </receive>
15   <assign name="assPrepId-bta-bt-PIP3A20"> <!-- prepare idReq call to backend  -->
16   <invoke partnerLink="ToBackend" operation="cbIdRequestOp"/> <!-- get id using
        synchronous call -->
17   <assign name="assPrepActChoice-bta-bt-PIP3A20"> <!-- prepare actChoice call to
        partner; choiceType is PartnerChoice  -->
18   <invoke partnerLink="ToPartner" operation="cbActivityChoiceOp"/>
19   <assign name="assPrepInit-bta-bt-PIP3A20"> <!-- prepare initAct call to partner
        -->
20   <invoke partnerLink="ToPartner" operation="cbInitActivityOp" />
21   <receive partnerLink="FromPartner" operation="cbInitAckOp">
```

```
   <!-- wait for initAck -->
   <correlations><!-- use corrCollaboration correlation set -->
   </receive>
  <assign name="assPrepActReady-bta-bt-PIP3A20"> <!-- prepare actReady call to backend -->
  <invoke partnerLink="ToBackend" operation="cbActivityReadyOp"/>
  <assign name="assPrepStart-bta-bt-PIP3A20"> <!-- prepare txStart call to BTA control process -->
   <!-- assign this process' InstanceIdentifier to BTA's ParentIdentifier -->
   <!-- assign the id requested from backend to BTA's InstanceIdentifier -->
  <invoke partnerLink="ToPIP3A20Requestor" operation="txStartOp"/>
  <receive partnerLink="FromPIP3A20Requestor" operation="txResultOp">
   <!-- use corrCollaboration correlation set -->
  <if name="switchState-bta-bt-PIP3A20-1">
   <condition>$txResult.../ns4:genericProtocolResult = 'AnyProtocolFailure' or
         $txResult.../ns4:genericProtocolResult = 'RequestReceiptFailure' or ...
   </condition> <!-- capture generic BT protocol result -->
   <assign name="route1-bta-bt-PIP3A20"><!-- Switch to next collaboration state -->
    <copy>
     <from>'techFail'</from>
     <to variable="cbState"/>
    </copy>
   </assign>
   <else> <!-- evaluate the business document exchanged; evaluation order is irrelevant -->
    <sequence name="switchStateEvalSeq1-bta-bt-PIP3A20">
     <assign name="assEval1-bta-bt-PIP3A20"> <!-- prepare call to XPath evaluation service -->
     <invoke partnerLink="ToXPath2Evaluator" operation="evaluateExpression"/>
     <if name="switchState-bta-bt-PIP3A20-2">
      <condition>$evalResp1-bta-bt-PIP3A20-FromXPath2Eval.xpath2EvaluationResponsePart</condition>
      <assign name="route2-bta-bt-PIP3A20"><!-- Switch to next collaboration state -->
       <copy>
        <from>'businessFail'</from>
        <to variable="cbState"/>
       </copy>
      </assign>
      <else>
       <sequence name="switchStateEvalSeq2-bta-bt-PIP3A20">
        <assign name="assEval2-bta-bt-PIP3A20"> <!-- prepare call to XPath evaluation service -->
        <invoke partnerLink="ToXPath2Evaluator" operation="evaluateExpression"/>
        <if name="switchState-bta-bt-PIP3A20-3">
         <condition>$evalResp2-bta-bt-PIP3A20-FromXPath2Eval.xpath2EvaluationResponsePart</condition>
         <assign name="route3-bta-bt-PIP3A20"><!-- Switch to next collaboration state -->
         <else> <!-- set of expressions must be disjoint AND complete -->
          <throw name="unknownResult-bta-bt-PIP3A20" faultName="cbFaults:UnknownResultException"/>
         </else>
        </if>
       </sequence>
      </else>
     </if>
    </sequence>
   </else>
  </if>
 </sequence>
</scope>
```

If the control process maps to the responder role of the BT, then BPEL code that is basically symmetric to listing 5.11 needs to be inserted. After setting up local variables, first the *actChoice* message of the partner control process is awaited. From a protocol perspective, this message is not strictly necessary, but it is included for easier generation of processes. Then, the *initAct* message of the partner control process is received and the backend is informed about the new BT execution using an *actReady* message. Subsequently, the BT responder control process is started using a *txStart* message. Again, the order of *actReady* and *txStart* message is aligned with the fact that the responder control process is supposed to send a message to the backend before the backend sends a message to the responder control process. Then, the start of the BT control process is acknowledged to the partner BC control process using an *initAck* message. After the result of the BT execution has been received from the responder control process, the evaluation of the result is performed exactly as in lines 33 to 71 of listing 5.10.

Listing 5.11: Initiating a BTA if Collaboration Role Maps to BTA Responder

```
<scope name="scope-bta-bt-PIP3A20">
 <variables><!-- provide local variables for implementation of micro protocol
     message exchanges -->
 <sequence name="seq-bta-bt-PIP3A20">
  <receive partnerLink="FromPartner" operation="cbActivityChoiceOp"> <!-- not
      strictly needed in this case -->
   <correlations> <!-- use corrCollaboration correlation set -->
  </receive>
  <receive partnerLink="FromPartner" operation="cbInitActivityOp">
   <correlations> <!-- use corrCollaboration correlation set -->
  </receive>
  <assign name="assPrepActReady-bta-bt-PIP3A20"> <!-- prepare actReady call to
      backend -->
   <!-- take over BTA instance identifier provided by partner -->
  <invoke partnerLink="ToBackend" operation="cbActivityReadyOp"/>
  <assign name="assPrepStart"> <!-- prepare txStart call to BTA responder process
      -->
  <invoke partnerLink="ToPIP3A20Responder" operation="txStartOp"/>
  <assign name="prepInitAck-bta-bt-PIP3A20"> <!-- prepare initAck call to partner
      -->
  <invoke partnerLink="ToPartner" operation="cbInitAckOp"/>
  <receive partnerLink="FromPIP3A20Responder" operation="txResultOp">
   <correlations> <!-- use corrCollaboration correlation set -->
  </receive>
  <if name="switchState-bta-bt-PIP3A20-1"> <!-- implement result evaluation;
      corresponds to requester mapping -->
 </sequence>
</scope>
```

Mapping Rule 5.3.3 (BCA)

This rule is applied if only one single BCA is eligible for execution and is the same as mapping rule 5.3.2 except for four differences. Firstly, the BCA is started immediately once it is enabled. Secondly, it is always the *leader* control process that initiates the BCA. Thirdly, first the lower level control processes are set up using a *cbStart* message and then the backends are informed about the BCA using an *actReady* message. This is different from the BTA strategy because at the collaboration level the first message exchanges are triggered by the backends and not by the control processes. Fourthly, the evaluation of BCAs is based on the end state labels of the corresponding ebBP-Reg model and not on ebBP ConditionGuardValues and expressions on business documents.

Mapping Rule 5.3.4 (Event-Based Choice)

The mapping rule for event-based choices depends on the distinction between *leader* and *non-leader* control processes. The basic idea is that the *leader* control process waits for either a backend request or a partner control process request and then starts the initiation of the requested activities. In case of concurrent requests, one out of two requests is selected in a non-deterministic manner. Conversely, the *non-leader* control process just relays the potential request of its backend and then awaits the decision of the *leader* control process. Note again that this is just a technical issue and does not affect business logic.

Listing 5.12 shows the structure of the *leader* mapping of an event-based choice (for the *'XOR-Fork'* of figure 4.11). After the definition of local variables, a BPEL `while` is executed until one of the event-based choice branches has been selected. In each iteration, a BPEL `pick` (starting in line 8) is used to choose from either a backend request or a partner request.

If a *reqAct* is received from the backend, a new scope (line 11) is installed. Within this scope, the validity of the requested activity is checked, i.e., the authorization of the backend to trigger the respective activity is evaluated. If the request is valid, *actChoice* messages are sent to the partner and to the backend to inform them about the selected branch of the event-based choice. Then, the `while` loop for repeating the event-based choice is broken by setting the corresponding boolean variable and the actual initialization of the selected activity is done in the next iteration of the global process `while` loop of listing 5.9. If the backend's request is not valid an *actChoice* message is prepared to be sent to the backend to reject the request later (see below). In order to support the different functions of the *actChoice* message, three different *choiceTypes* are included, *'Accept'* for accepting an activity request, *'PartnerChoice'* for transmitting the name of an activity chosen by the partner, and *'Invalid'* for invalid requests. Note that the scope starting in line 11 also defines an `eventHandler` for consuming *actReq* messages of the partner control process (lines 13 to 18) that is used to store the partner's request. This scope is necessary to avoid the situation that the partner tries to deliver an *actReq* message while the focal control process tries to deliver an *actChoice* message (remember that the communication style is synchronous). As the underlying Web Services calls for delivering the messages eventually would time out, the mutual blocking would not constitute a classical deadlock, but the `eventHandler` is installed to avoid unnecessary waiting times.

The processing of the second `onMessage` branch of the `pick` for consuming a partner request is symmetric.

After the `pick` has been processed, the determination of an event-based choice branch is evaluated. If no branch has been chosen yet and if a backend or partner request is still stored then these stored requests are processed. There are basically two instantiations of this situation. Either the backend delivered an invalid *reqAct* message first and the partner *reqAct* was received while performing the backend's request or the other way round. Therefore, the BPEL `if` constructs in lines 39 and 57 are used to cover those two cases. The BPEL code for doing this is essentially the same as for the `onMessage` branches of the event-based choice `pick`, but no `eventHandlers` are used to consume concurrent requests. This is not necessary, because the partner control process and backend are still waiting for an *actChoice* message. For the `if` construct starting in line 39, this is the case because the invalid request of the backend was not rejected immediately, but only after the potential concurrent request of the partner has been evaluated (cf. above). If both requests, the backend request and the partner request, are invalid then the `if` structures in lines 58 and 62 are used to deliver the missing *actChoice* messages and the next iteration of the event-based choice loop waits for the next requests.

Listing 5.12: Mapping of an Event-Based Choice for a Leader Control Process

```
1  <scope name="scope-fork-continue">
2   <variables> <!-- provide local variables for implementation of micro protocol
        message exchanges -->
3   <sequence name="seq-fork-continue">
4    <assign name="initVar-fork-continue"> <!-- some initializations -->
5    <while name="while-fork-continue">
6     <condition>not($decided-fork-continue)</condition> <!-- while no branch
          selected -->
7     <sequence name="seqWhile-fork-continue">
8      <pick name="mPick-fork-continue">
9       <onMessage partnerLink="FromBackend" operation="cbRequestActivityOp">
10       <correlations> <!-- use corrCollaboration correlation set -->
11       <scope name="scopeBeNeed-fork-continue">
12        <eventHandlers>
13         <onEvent partnerLink="FromPartner" operation="cbRequestActivityOp">
14          <correlations> <!-- use corrCollaboration correlation set -->
15          <scope name="scopeLatePartner-fork-continue">
16           <assign name="assStoreLatePartner-fork-continue"> <!-- store partner
                request -->
17          </scope>
18         </onEvent>
19        </eventHandlers>
20        <if name="ifValidReqBe-fork-continue">
21         <condition>'bta-bt-PIP3A23' = $reqAct.../ns4:activityName</condition> <!--
              if valid request -->
22          <sequence name="seqBeNeed-fork-continue">
23           <assign name="assPrepBeNeed-fork-continue"> <!-- prepare actChoice calls
                 to backend and partner -->
24           <invoke partnerLink="ToPartner" operation="cbActivityChoiceOp"/>
25           <invoke partnerLink="ToBackend" operation="cbActivityChoiceOp"/>
26           <assign name="switchStateBeNeed-fork-continue"> <!-- break ebc while and
                 switch to preparing selected activity -->
27          </sequence>
28         <else>
29          <sequence name="seqBeInvalid-fork-continue">
30           <assign name="assBeNeedInvalid-fork-continue"> <!-- prepare actChoice
                call to backend -->
31          </sequence>
32         </else>
33        </if>
34       </scope>
35      </onMessage>
36      <onMessage partnerLink="FromPartner" operation="cbRequestActivityOp">
37        <!-- symmetric to onMessage for backend reqAct -->
38     </pick>
39     <if name="ifLatePa-fork-continue">
40      <condition>not($decided-fork-continue) and $concPaNeed-fork-continue</
          condition>
41      <if name="ifValidReqPaLate-fork-continue">
42       <condition>'bta-bt-PIP3A21' = $reqAct.../ns4:activityName or 'bta-bt-PIP3C3'
            = $reqAct.../ns4:activityName</condition>
43       <sequence name="seqPaNeedLate-fork-continue">
44        <assign name="assPrepPaNeedLate-fork-continue"> <!-- prepare actChoice
            calls to backend and partner -->
45        <invoke partnerLink="ToBackend" operation="cbActivityChoiceOp"/>
46        <invoke partnerLink="ToPartner" operation="cbActivityChoiceOp"/>
47        <assign name="switchStatePaNeedLate-fork-continue"> <!-- break ebc while
            and switch to preparing selected activity -->
48       </sequence>
49       <else>
50        <sequence name="seqPaInvalidLate-fork-continue"> <!-- in case of invalid
            request -->
```

```
51         <assign name="assPaNeedInvalidLate-fork-continue"> <!-- prepare actChoice
               call to partner -->
52         <invoke partnerLink="ToPartner" operation="cbActivityChoiceOp"/>
53        </sequence>
54       </else>
55      </if>
56     </if>
57     <if name="ifLateBe-fork-continue"> <!-- symmetric to ifLatePa-fork-continue --
           >
58     <if name="ifBeInvalid-fork-continue">
59      <condition>not($decided-fork-continue) and $beInvalid-fork-continue</
            condition>
60      <invoke partnerLink="ToBackend" operation="cbActivityChoiceOp"/>
61     </if>
62     <if name="ifPaInvalid-fork-continue"> <!-- symmetric to ifBeInvalid-fork-
           continue -->
63     <assign name="assClearConcurRequests"> <!-- clean up for next iteration -->
64    </sequence>
65   </while>
66  </sequence>
67 </scope>
```

The code for the *non-leader* implementation of an event-based choice given in listing 5.13 is much simpler compared to the *leader* mapping. Basically, another `while` loop (line 5) is used to iteratively wait for an *actChoice* message of the partner control process or a *reqAct* message of the backend using a BPEL `pick` (line 8). In case a *reqAct* message from the backend is received (BPEL `onMessage` construct starting in line 9) this message is relayed to the *leader* control process and then the *leader's actChoice* message is awaited. The type of the actChoice message may be either *'Invalid'* if the backend's request was invalid, *'Accept'* if the backend's request was granted or *'PartnerChoice'* if the *leader's* backend requested an activity slightly before the *non-leader's* backend. In the latter case the `onEvent` construct starting in line 13 of listing 5.12 would have been used to consume the *reqAct* relayed by the *non-leader* control process. In any case, the contents of the *leader's actChoice* is sent to the backend and if the type is not *'Invalid'* (checked in the `if` structure starting in line 19) then the event-based choice `while` is left.

The second `onMessage` of the non-leader's `pick` starting in line 25 consumes an unsolicited *actChoice* of the leader control process that corresponds to a granted *reqAct* message of the leader's backend. This *actChoice* is relayed to the backend and then the `while` for controlling the event-based choice is left. A check for the *actChoice* type is left out because the *'Invalid'* type is only possible if the *non-leader* control process transmitted a *reqAct* beforehand. Note that an `eventHandler` for consuming a potential *reqAct* from the non-leader backend while processing the *actChoice* of the leader control process is installed (starting in line 29) in order to resolve concurrent requests.

Listing 5.13: Mapping of an Event-Based Choice for a Non-Leader Control Process

```
1 <scope name="scope-fork-continue">
2  <variables> <!-- provide local variables for implementation of micro protocol
       message exchanges -->
3  <sequence name="seq-fork-continue">
4   <assign name="initVar-fork-continue"> <!-- set while variable -->
5   <while name="while-fork-continue">
6    <condition>not($decided-fork-continue)</condition>
7    <sequence name="seqWhile-fork-continue">
8     <pick name="pick-fork-continue">
9      <onMessage partnerLink="FromBackend" operation="cbRequestActivityOp">
```

```
      <correlations> <!-- use corrCollaboration correlation set -->
      <sequence name="seqReqAct-fork-continue-BE">
       <assign name="prep-reqAct-fork-continue-ToPartner"> <!-- prepare reqAct
           call to partner -->
       <invoke partnerLink="ToPartner" operation="cbRequestActivityOp"/>
       <receive partnerLink="FromPartner" operation="cbActivityChoiceOp">
        <correlations> <!-- use corrCollaboration correlation set -->
       </receive>
       <assign name="prep-actChoice-fork-continue-ToBackend"> <!-- prepare
           actChoice call to backend -->
       <invoke partnerLink="ToBackend" operation="cbActivityChoiceOp"/>
       <if name="decContinue-actChoice-fork-continue"> <!-- check type of
           actChoice -->
        <condition>'PartnerChoice' = $actChoice.../ns4:choiceType or 'Accept' = $
            actChoice.../ns4:choiceType</condition>
        <assign name="SetDone-actChoice-fork-continue"> <!-- break ebc while and
            switch to preparing selected activity -->
       </if>
      </sequence>
     </onMessage>
     <onMessage partnerLink="FromPartner" operation="cbActivityChoiceOp">
      <correlations> <!-- use corrCollaboration correlation set -->
      <scope name="scopeActChoice-fork-continue">
       <eventHandlers>
        <onEvent partnerLink="FromBackend" operation="cbRequestActivityOp">
         <correlations> <!-- use corrCollaboration correlation set -->
         <scope name="actChoiceUnSol-resolveBECall">
          <empty name="resolveCall"/> <!-- just forget this reqAct; the partner's
              actChoice will resolve -->
         </scope>
        </onEvent>
       </eventHandlers>
       <sequence name="seqActChoice-fork-continue">
        <assign name="prep-actChoiceUnSol-fork-continue-ToBackend"> <!-- prepare
            actChoice call to backend -->
        <invoke partnerLink="ToBackend" operation="cbActivityChoiceOp"/>
        <assign name="SetDone-actChoiceUnSol-fork-continue"> <!-- break ebc while
            and switch to preparing selected activity -->
       </sequence>
      </scope>
     </onMessage>
    </pick>
   </sequence>
  </while>
 </sequence>
</scope>
```

Two simplifications of the event-based choice mapping are possible depending on the type of outgoing activities that can be triggered. If all BTAs the event-based choice links to map the focal control process' role to the BT requesting role and there is no link to an BCA then all that needs to be done is waiting for a *reqAct* of the backend, validate it and forward it as an *actChoice* to the partner control process. Conversely, if all BTAs the event-based choice links to map the focal control process' role to the BT responding role then the mapping consists of waiting for the partner control process' *actChoice* and forwarding it to the backend. Note that, in both cases, the initiation of the respective activities is performed by analogy with mapping rule 5.3.2 in the next iteration of the process level `while` loop.

Mapping Rule 5.3.5 (Parallel)

For the mapping of a parallel structure remember that each branch consists of exactly one BCA and that no interaction between the branches is defined. Moreover, the mapping rule for a parallel structure depends on the distinction between *leader* and *non-leader* control processes. Listing 5.14 shows the mapping of the *leader* control process that starts initiating all the connected BCAs once the parallel structure has been selected in the process level `while` loop described in listing 5.9. For the use case of figure 4.11 two instances of a collaboration centered around PIP 3A19 are performed. Apparently, the mapping does not consist of one single BPEL `flow` but starts out with a series of micro protocol message exchanges organized in a sequential structure. The reason for this is that the *actChoice*, *initAct* and *initAck* messages have to be correlated with the partner control process instance. Hence the BPEL process correlation mechanism is used for process identification and as the `partnerLink`, `portType`, `operation` and `correlationSet` used for the exchange of two micro protocol messages of the same type are identical, enabling such receive events concurrently in a BPEL `flow` would result in a BPEL `conflictingReceive` error (cf. [137, section 10.4]). Consistently, the *initAck* messages for acknowledging the setup of the lower level BC control processes by the partner control process are received in sequence (lines 13 and 16 of listing 5.14). Note that initiating the BCAs of a parallel structure does not violate the ebBP-Reg execution semantics because there is no order imposed on the branches of a parallel structure. In addition, the time for exchanging micro protocol messages typically is a matter of seconds whereas performing BCs typically takes hours to days due to human involvement. Once the *initAck* messages for all branches have been received from the partner control process, the child BC control processes are set up within a BPEL `flow` structure where each BPEL `sequence` within the `flow` constitutes a BPEL level parallel branch (starting in lines 20 and 25). In each of these branches, a *cbStart* message is used to create a new child BC control process the creation of which is then signaled to the backend using *actReady* messages. In case the backend is not able to deal with concurrently receiving two *actReady* messages then the initiation of the child BC control processes could be performed in sequence as well for the reasons just explained. Finally, another BPEL `flow` (starting in line 31) is used to collect the results of the child collaborations. In this case, the use of concurrently active branches for collecting results is strictly required because the sequence of result messages from the child BC processes is not known in advance. However, as different `partnerLinks` and `correlationSets` are used for receiving these messages, using the same operation does not do any harm (cf. lines 33 and 38). The mapping of a *non-leader* control process is symmetric and omitted for reasons of conciseness. The basic structure of initiating an activity that is necessary for creating the *non-leader* process mapping is shown in mapping rule 5.3.2.

Listing 5.14: Mapping of a Parallel Structure for a Leader Control Process

```
1 <scope name="scope-fk-parallel">
2  <variables> <!-- provide local variables for implementation of micro protocol
       message exchanges -->
3  <sequence name="seq-fk-parallel"> <!-- coordinate execution of BCAs -->
4   <assign name="assPrepIds-fk-parallel"> <!-- prepare idReq calls to backend -->
5   <invoke name="invGetID-ba-bc-single-3A19-1" partnerLink="ToBackend" operation="
       cbIdRequestOp"/>
6   <invoke name="invGetID-ba-bc-single-3A19-2" partnerLink="ToBackend" operation="
       cbIdRequestOp"/>
```

```
  <assign name="assPrepActChoice-fk-parallel"> <!-- prepare actChoice calls to
      partner -->
  <invoke name="invActChoice-ba-bc-single-3A19-1" partnerLink="ToPartner"
      operation="cbActivityChoiceOp"/>
  <invoke name="invActChoice-ba-bc-single-3A19-2" partnerLink="ToPartner"
      operation="cbActivityChoiceOp"/>
  <assign name="assPrepInit-fk-parallel"> <!-- prepare initAct calls to partner --
      >
  <invoke name="invInit-ba-bc-single-3A19-1" partnerLink="ToPartner" operation="
      cbInitActivityOp">
  <invoke name="invInit-ba-bc-single-3A19-2" partnerLink="ToPartner" operation="
      cbInitActivityOp">
  <receive name="recInitAck-ba-bc-single-3A19-1" partnerLink="FromPartner"
      operation="cbInitAckOp">
   <correlations> <!-- use corrCollaboration correlation set -->
  </receive>
  <receive name="rec-ba-bc-single-3A19-2-FromPartner" partnerLink="FromPartner"
      operation="cbInitAckOp">
   <correlations> <!-- use corrCollaboration correlation set -->
  </receive>
  <flow name="flow1-fk-parallel"> <!-- start the child control processes -->
   <sequence name="seq-ba-bc-single-3A19-1">
    <assign name="assPrep-ba-bc-single-3A19-1"> <!-- prepare cbStart/actReady
        calls to child control processes/backend -->
    <invoke name="invStart-ba-bc-single-3A19-1" partnerLink="ToBcSingle3A191"
        operation="cbStartOp">
    <invoke name="invActReady-ba-bc-single-3A19-1" partnerLink="ToBackend"
        operation="cbActivityReadyOp">
   </sequence>
   <sequence name="seq-ba-bc-single-3A19-2">
    <assign name="assPrep-ba-bc-single-3A19-2"> <!-- prepare cbStart/actReady
        calls to child control processes/backend -->
    <invoke name="invStart-ba-bc-single-3A19-2" partnerLink="ToBcSingle3A192"
        operation="cbStartOp">
    <invoke name="invActReady-ba-bc-single-3A19-2" partnerLink="ToBackend"
        operation="cbActivityReadyOp">
   </sequence>
  </flow>
  <flow name="flow2-fk-parallel"> <!-- collect child collaboration results -->
   <sequence name="seq-collect-ba-bc-single-3A19-1">
    <receive name="collect-ba-bc-single-3A19-1" partnerLink="FromBcSingle3A191"
        operation="bc-single-3A19CbResultOp">
     <correlations> <!-- use corrCollaboration correlation set -->
    </receive>
   </sequence>
   <sequence name="seq-collect-ba-bc-single-3A19-2">
    <receive name="collect-ba-bc-single-3A19-2" partnerLink="FromBcSingle3A192"
        operation="bc-single-3A19CbResultOp">
     <correlations>  <!-- use corrCollaboration correlation set -->
    </receive>
   </sequence>
  </flow>
  <assign name="switchState-fk-parallel"> <!-- switch to next collaboration state
      -->
 </sequence>
</scope>
```

Mapping Rule 5.3.6 (Terminal)

The last rule describes the mapping of an ebBP-Reg terminal state and does not depend on the *leader/non-leader* distinction. Listing 5.15 shows the mapping of the terminal state *'businessSuccess'* of figure 4.11 that consists of a simple assignment for breaking the process level `while` loop. The actual propagation of the result then is done in a separate scope that follows the process level `while` loop which is shown in listing 5.16. In this scope, it is first checked whether the control process has been interrupted (line 8 of listing 5.16) due to a `TimeToPerform` timeout (cf. discussion of listing 5.9). If so, a technical error is reported to the backend and the value of the *'cbState'* variable is reported otherwise.

Listing 5.15: Mapping of a Terminal State

```
<!-- breaks the main loop 'cbWhile' -->
<if name="cbSwitch-businessSuccess">
 <condition>$cbState = 'businessSuccess'</condition>
 <sequence name="Seq-businessSuccess">
  <assign name="Ass-businessSuccess">
   <copy>
    <from>true()</from>
    <to variable="inEndState"/>
   </copy>
  </assign>
 </sequence>
</if>
```

Listing 5.16: Wrap up Code of a BC control process

```
<!-- propagates the results to the backend in a final scope-->
<scope name="FinalScope">
 <variables>
  <variable name="bcResult-ToBackend"/> <!-- result variable -->
 </variables>
 <sequence name="FinalSeq">
  <if name="FinalIf">
   <condition>$interrupted</condition>
   <assign name="prepReply">
    <copy>
     <from>$CbStartOpIn.../ns3:CollaborationMetaBlock</from>
     <to>$bcResult-ToBackend.../ns3:CollaborationMetaBlock</to>
    </copy>
    <copy>
     <from>'techFail'</from>
     <to>$bcResult-ToBackend.../ns2:bc-controlFlowTestResult</to>
    </copy>
   </assign>
   <else>
    <assign name="prepReplyReg"> <!-- copy cbState value into reply message -->
   </else>
  </if>
  <invoke partnerLink="ToBackend" operation="bc-controlFlowTestCbResultOp">
 </sequence>
</scope>
```

5.4. Chapter Summary

This chapter demonstrated the implementation of ebBP BTs and BCs using Web Services and BPEL technology. The implementation of BTs as core building block for B2Bi choreographies is covered in section 5.2 whereas the implementation of ebBP-Reg as the more demanding bilateral choreography model is described in section 5.3. The underlying assumptions for using Web Services communication technology (such as implementability of mutual authentication) are extensively discussed and validated. In addition, the conceptual benefits of choosing Web Services and BPEL as well as the flexible integration architecture based on the *control process concept* are highlighted.

Although the actual purpose of the implementation concept for BTs and BCs is the validation of the B2Bi choreography styles and the associated execution semantics, deriving a production level framework for automatically deriving BT and BC control processes is highly desirable. The implementation template for BTs and the BPEL mapping rules for BCs provide the foundation and reference for deriving such tooling. In that regard, note that a subset of commonly available BPEL elements is used which simplifies supporting multiple BPEL platforms. The sanity of the given artifacts is proven by means of prototypic implementations for the GlassFish-openESB platform (cf. section 5.1). In addition, a formal validation for the BT execution model is given in appendix D. Note further that some joint work was carried out during this dissertation project that demonstrates the feasibility of implementing such translation tooling at both, the BT level [54] and the BC level [160, 182]. The work published in [160, 182] further provides guidelines for implementing ebBP-ST.

Yet, the implementation of a production level ebBP-BPEL tool chain would require considerable effort. Support for BPEL elements varies from BPEL engine to BPEL engine and there are considerable interoperability problems in applying policy-based WS-* features [183]. Furthermore, the variety of technical configuration options for communication platforms would have to be taken into account. These factors taken together with the need of testing the integration with the large variety of internal business applications used in the B2Bi domain call for a concerted project of major B2Bi adopters and communities such as RosettaNet or EDIFICE. However, the benefits of adopting the control process-based integration architecture (cf. introduction of this chapter) and providing a framework with unambiguous execution semantics seem to justify such an effort. The separation of control flow and business logic enables automatically deriving fully executable process definitions. Moreover, the modular structure of control processes facilitates the use of traditional B2Bi technologies like EDI and AS2 at the BT level for supporting high-volume data. But most significantly, B2Bi partners would regain control over the cross-organizational processes and would not have to rely blindly on the service of B2Bi solution vendors.

This chapter evaluated the sanity of the proposed B2Bi styles at the implementation level and gave guidance for implementers. The next chapter will show a visualization of these B2Bi styles that abstracts from ebBP details and serves as communication tool for the business perspective.

6. Visualizing B2Bi Choreographies

The contents of this chapter have been produced by the author during his participation in the RosettaNet MCC effort[1] with the target of providing a choreography format for specifying complex B2Bi interactions based on RosettaNet PIPs. Due to the tight conceptual relationship between RosettaNet PIPs and ebBP BusinessTransactions as well as the results of MCC phase 1 [172] that specifies how to implement PIPs using Web Services such that mutual agreement is guaranteed, the applicability of ebBP-ST, ebBP-Reg and SeqMP is given. The visualizations of those B2Bi choreography styles produced during this work eventually got accepted by the RosettaNet MCC team and were contributed to the *RosettaNet Methodology for Creating Choreographies* standard [173]. The author is deeply grateful and indebted to the RosettaNet MCC team for all the support, feedback and discussion during creating the BPMN visualization. The contents below exclusively are based on the author's own contributions to [173].

Technically speaking, the specification of B2Bi choreographies using ebBP is sufficient to precisely capture the business document exchanges of integration partners and to derive implementation artifacts from that. However, good practices of design science demand for communication of research results to non-technical audiences as well (cf. section 1.3). The visualization of ebBP-Reg, ebBP-ST and SeqMP using a compact notation helps in making the results of this work accessible to non-technical audiences. At the same time, the specification of ebBP dialects alone would have raised questions with respect to the suitability of this work because ebBP is a rather verbose and bulky format for choreography specification. The BPMN visualization provided here demonstrates that the B2Bi choreography styles identified can be embedded into B2Bi development approaches that leverage visual abstractions as input models.

The choice of BPMN as visual choreography notation is motivated by the desire of representing complete choreography definitions in one single diagram on the one hand and using an international standard on the other hand. Other choreography notations either require the definition of multiple views for choreography specification (UMM [208]), or are not a standard (BCL [248], BPEL4Chor [31], Let's Dance [245]). However, this does not mean that these notations are not usable for B2Bi choreography definition at all. In particular, the multi-view approach of UMM is promising when it comes to establishing a comprehensive framework that includes tasks such as the definition of business documents. However, this task is not in the scope of the work

[1] `http://www.rosettanet.org/dnn_rose/Standards/RosettaNetPrograms/FoundationalPrograms/ActiveFoundationalPrograms/MessageControlChoreography/tabid/3096/Default.aspx`, last access: 12/20/2011

at hand. Furthermore, the BPMN visualization contained in the ebBP standard itself is not applicable to this work as it is incomplete in terms of grammar rules, outdated in using version 1.0 of BPMN, and inadequate in defining an interconnection choreography style instead of interaction choreography style (see section 2.3) as ebBP does.

The BPMN visualization here concentrates on strict choreographies (instead of cartography choreographies) as discussed in chapter 4 and therefore reflects the design of ebBP-ST, ebBP-Reg and SeqMP. In addition, building the visualization upon these B2Bi choreography styles helps in selecting the modeling concepts that are relevant for the most common B2Bi scenarios from the huge wealth of BPMN functionality and in ensuring that the corresponding visual models have unambiguous semantics. Note that the *RosettaNet Methodology for Creating Choreographies* also allows for using the full set of BPMN functionality to specify cartography choreographies of arbitrary complexity and meaning. However, this is not relevant for the scope of this work.

For B2Bi choreography visualization, BPMN is restricted and amended in two ways. Firstly, the set of BPMN elements is reduced to those that are needed for representing B2Bi choreographies. Secondly, the rules for connecting those elements such that the choreography styles of chapter 4 are representable are given. For both parts, special emphasis is put on accessibility to a non-technical audience while maintaining technical completeness and soundness as far as possible. For the definition of grammar rules, the formal representations of chapter 4 are replaced with visual examples that demonstrate legal uses of concepts. However, the visualization of all illegal uses is not given in order to keep complexity low.

For technical validation, several use cases of RosettaNet's *RIG* library as well as *eBusiness Process Scenario* library have been used that leverage the different control flow constructs of this work's B2Bi choreography styles. In addition, the corresponding BPMN models manually have been translated into ebBP models and the technical detail that needs to be filled in for defining complete ebBP models has been identified. These resources are available as part of the *RosettaNet Methodology for Creating Choreographies* deliverables[2]. Validation in terms of usability is provided by submitting the deliverables to the RosettaNet MCC team that also comprised non-technical oriented members. Indeed, some modeling concepts such as a new visual element for representing shared states were given up due to missing support in the MCC team. However, it is vital to note (again) that usability is not the core goal of this work and therefore an elaborate scientific framework for assessing usability has not been leveraged.

Taken the scientific underpinning of B2Bi choreography styles together with the validation by means of real-world use cases, the adaptation of BPMN to represent

[2]`http://www.rosettanet.org/dnn_rose/DocumentLibrary/tabid/2979/DMXModule/624/Command/Core_Download/Method/attachment/Default.aspx?EntryId=9858`, last access: 12/20/2011

these choreography styles as presented in this chapter can be said to define a kind of B2Bi choreography profile of BPMN.

The remainder of this chapter is organized as follows. Section 6.1 describes the selection of BPMN elements and how these are used for representing ebBP elements. The composition of those elements in a manner such that this work's choreography styles can be represented then is described in section 6.2. Finally, section 6.3 discusses in how far the contents of this chapter comply with the BPMN standard, the suitability of BPMN for B2Bi choreography modeling, and the technical gap between BPMN models and corresponding ebBP models.

6.1. Selection of BPMN Elements

This section describes the *"BPMN Choreography"* [150, section 11] constructs that are used for representing B2Bi choreographies using a series of so-called "Construct Advices". Although some samples in this section will comprise more than one construct, the actual interplay of constructs as well as composition rules are contained in section 6.2. In particular, constraints on the use of elements or names of elements may be tightened for ensuring executability or analyzability.

It is essential to note that these advices have been created for the *RosettaNet Methodology for Creating Choreographies* standard where PIPs are used as atomic building blocks instead of ebBP BTs. For the purpose of visualization as presented below, the terms *"PIP"* and *"BT"* can be used interchangeably. This is particularly true because a protocol for performing PIPs using Web Services technology such that mutual agreement between integration partners is ensured has been contributed to RosettaNet MCC Web Services profile [172]. Therefore, the figures are created using RosettaNet PIP terminology, but the descriptions refer to ebBP BTs.

Construct Advice 6.1.1 (Representing BusinessTransactions)

An execution model for ebBP BTs based on visual communicating state machines has been defined in section 4.3. As the number of BT configurations is finite and as BT configurations focus on technical detail it is not sensible to remodel that execution model in BPMN. Instead, BTs are modeled as atomic building blocks and only those parts of the BT configuration that are relevant on a business level are visually represented. The most important aspects to be represented for a BT within a choreography are its type, its activity id as well as the role assignment.

The BT type identifies a particular BT configuration and its name can be defined as needed in ebBP. Therefore, the label for the BT type must comply with those naming rules. If RosettaNet PIPs are used, the type corresponds to the PIP identifier as defined in RosettaNet's cluster/segment/PIP taxonomy (cf. section 2.4). The activity id is a name for identifying a particular BTA that represents the execution of a particular BT configuration at a particular control flow point within the choreography. In addition, the role mapping as well as an upper time limit can be defined for a BTA. The activity id is important for distinguishing between several uses of the same

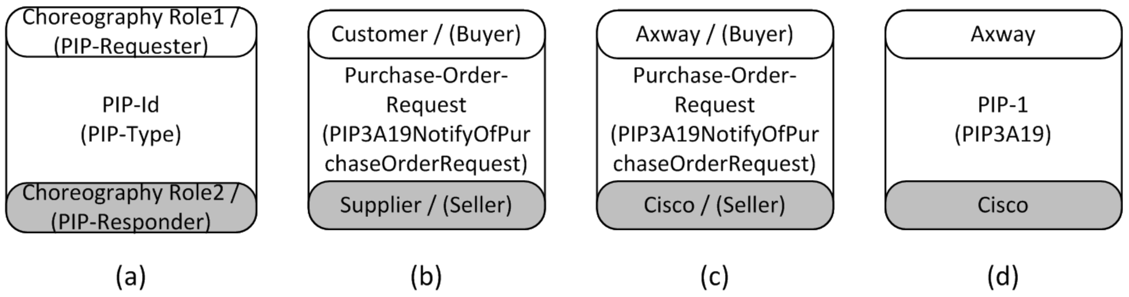

Choreography Role1 / (PIP-Requester)
PIP-Id (PIP-Type)
Choreography Role2 / (PIP-Responder)
(a)
Customer / (Buyer)
Purchase-Order-Request (PIP3A19NotifyOfPurchaseOrderRequest)
Supplier / (Seller)
(b)
Axway / (Buyer)
Purchase-Order-Request (PIP3A19NotifyOfPurchaseOrderRequest)
Cisco / (Seller)
(c)
Axway
PIP-1 (PIP3A19)
Cisco
(d)

Figure 6.1.: Sample PIP Representations

BT type within the same choreography. For example, the exchange of a purchase order (PIP 3A19) may be used for different purposes (placing a purchase order to be answered within 3 days and placing an additional order to be answered within 12 hours) within the same choreography. In this situation, the activity id helps in distinguishing the two different uses. Finally, the mapping of the choreography roles to the BT roles must be defined, i.e., it must be clear which of the choreography roles is assigned the requester role or responder role of the BT, respectively.

For the following explanations, note that a PIP definition distinguishes the *communication role* (requester or responder) from the *functional role* (say, Buyer or Seller). Whereas the communication role determines the sender and receiver of the business document, respectively, the functional role characterizes the business meaning of dealing with the document (the Buyer commits to buying the items listed in the business document whereas the Seller commits to delivering the items). In the RosettaNet methodology, there is a one-to-one relationship between the communication roles and the functional roles of a PIP, i.e., a functional role is assigned either the requester role or the responding role. Note that statically defining communication role and functional role is different from the ebBP conception where only the communication role is predefined. As ebBP allows for giving alternative names to the requester and responder role, i.e., defining a use case specific functional role, capturing PIP functional roles is possible nonetheless.

Figure 6.1 (a) shows an abstract characterization of how to represent a BT as a BPMN 2.0 *choreography task*. The center of the rounded rectangle carries the activity id followed by the actual type of the BT in parentheses (in this case a PIP type identifier). The two choreography roles that perform a BT are listed within two bands at the top and at the bottom of the rectangle. The functional roles that the choreography roles play can be listed after a slash character within parentheses. The communication role that a choreography role takes is specified by coloring the bands. The white band identifies the requester role whereas the gray band identifies the responder role. The position of the bands is insignificant, i.e., you could have a gray band at the top and a white band at the bottom as well. As there is a one-to-one relationship between functional roles and communication roles (either statically defined by RosettaNet PIPs, or defined by the ebBP modeler), the mapping to functional roles within the bands can be omitted. For example, look at Figure 6.1

(b) that exemplifies the specification of performing PIP 3A19. The choreography role 'Customer' is mapped to the PIP functional role 'Buyer'. But, as the Buyer role is defined to take the requester role in the definition of PIP 3A19 and as the Customer role is assigned the requester role in Figure 6.1 (b) by being put into the white band, it is clear that the Customer role takes the Buyer role. Hence, the labels *"Customer / (Buyer)"* and *"Supplier / (Seller)"* in figure 6.1 (b) could be reduced to the labels *"Customer"* and *"Supplier"* without losing information.

Figure 6.1 (c) and (d) express the exact same information as figure 6.1 (b). The only difference is the style of specification. Instead of "anonymous" choreography roles such as Customer and Supplier, "scenario specific" roles such as 'Axway' and 'Cisco' are used. From a technical perspective, this does not make a difference. If 'Software AG' was to interact with 'Cisco' then Software AG could either be declared to take the Customer role or the Axway role. From a modeling perspective, however, assigning Software AG the Axway role may be confusing for the reader. Similarly, there is no technical reason why choreography role names should be required to be distinct from PIP role names. It is perfectly acceptable to map a choreography Seller role to a PIP Seller role.

For the definition of activity ids, any string that does not contain special characters is admissible. In addition, activity ids must be unique relative to the enclosing choreography (in order to ensure translatability of ebBP choreographies, cf. section 6.3).

If PIPs are used, additional restrictions in terms of naming must be respected (cf. [173]).

Construct Advice 6.1.2 (Representing Start States)

Start states represent the entry points into choreographies and are visualized using BPMN *Start Event* nodes as depicted in figure 6.2. By following the outgoing transitions of start states the initially admissible BTs of a particular choreography can be identified.

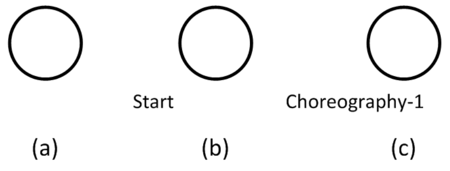


Figure 6.2.: Sample Start States

Note that it is advisable to assign a name to start states that begins with 'Start' for better readability although neither the use of a name nor a particular format is necessary for defining valid models. Therefore (a), (b) and (c) of figure 6.2 all are acceptable. The only limitation is that special characters must not be used.

Construct Advice 6.1.3 (Representing End States)

End states represent the termination of choreographies and may indicate different outcomes. End states as used in this work do not express any actions, but just define that no more BTs will be performed. Therefore, basic BPMN *End Events* are used for representing end states. End states can be distinguished from start states by the following two properties:

- In conformance to the BPMN specification, end states have *'thick'* lines [150, section 11.5.3] whereas start states have *'thin'* lines.
- End states only have incoming transitions whereas start states only have outgoing transitions.

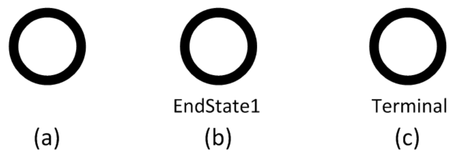


Figure 6.3.: Sample End States

There are no naming rules for end states except that special characters must not be used. Moreover, end state ids must be unique relative to the enclosing choreography. Hence, the labels in figure 6.3 (a) - (c) all are acceptable end state representations.

Construct Advice 6.1.4 (Representing Transitions)

Transitions connect the various states or nodes of choreographies and are crucial for specifying control flow. The interpretation of transitions is essential for defining the semantics of a visual language. The semantics defined in chapter 4 are adopted for specifying control flow. Therefore, *Sequence Flows* as defined in the BPMN standard are used for connecting the various choreography elements, but the way transitions are represented does not fully comply with BPMN general considerations [150, section 8.3.13] and choreography considerations [150, section 11.3.1] for Sequence Flows. The details of interpreting transitions are formally defined in chapter 4 and informally presented in the next section whereas the visual representations of different transition types are described here.

Figure 6.4 shows the four basic types of transitions that are used in this work. All types of transitions are directed which implicitly means that the source state is left and that the target state is entered upon "firing" the transition. A transition of type (a) may be fired when the subsequent BT or choreography is started, when a related timer runs out or when leaving any state is to be specified without elaborating on the conditions for moving from one state to another. Transitions of type (b) and (c) carry condition expressions that are used to evaluate the result of a BT execution. That means that a transition only can fire if the expression evaluates to true. The

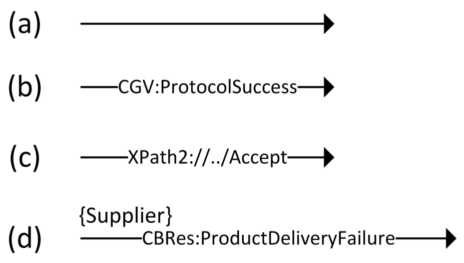


Figure 6.4.: Types of Transitions

definition of a condition expression consists of the name of the expression language separated from the actual expression by means of a colon. Either the ebBP CGV language is used to capture generic protocol outcomes such as ‘AnyProtocolFailure’ or ‘ProtocolSuccess’ or expression languages for evaluating the contents of the exchanged business documents are used. Note that evaluating the contents of a business document only is valid if the BT succeeded from a protocol perspective. Following the concept outlined in ebBP, XPath2 is suggested as evaluation language of XML based content. However, as Figure 6.4 (c) indicates, XPath2 expressions may be way too complex to be included in a visual model. Therefore, the problem of visualizing valid and complete XPath2 expressions is left to tool implementations. Apart from CGV and XPath2, the use of alternative expression languages can be agreed upon by integration partners.

The CBRes (short for CollaborationResult) language consistently is used for evaluating the result of choreographies (or collaborations in terms of ebBP and UMM). The admissible results of a choreography are defined to be the names of the end states of the choreography. Transitions that evaluate the result of a choreography additionally may carry an escalation set definition in curly braces (as defined in section 4.6).

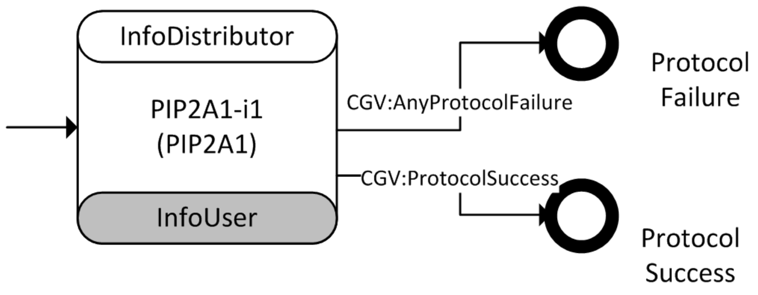


Figure 6.5.: Sample Usage of Transitions

Finally, this work leaves it open to integration partners to define project specific labels. For example, the Trigger-Guard-Effect notation as defined in the UML standard [149, section 15.3.14] (sometimes referred to as Event-Condition-Action notation) could be agreed upon by the partners.

Figure 6.5 shows a basic example of using transitions. The incoming transition on the left-hand side of the PIP 2A1 choreography task is fired when the 'InfoDistributor' triggers the PIP (as the InfoDistributor is assigned the "initiator band" of the task). On the right-hand side, two transitions are used to evaluate the PIP result using the CGV expressions 'AnyProtocolFailure' or 'ProtocolSuccess'. The determination of such CGV expressions has been defined in [172]. Depending on the result of PIP 2A1, either the end state called 'ProtocolFailure' or the end state 'ProtocolSuccess' is entered upon completion of the BT execution protocol. Note that both choreography roles follow the same transition as the result of a BT is always synchronized by means of the execution model of section 4.3.

Construct Advice 6.1.5 (Representing Choreographies)

The BPMN standard does not offer a dedicated construct for visualizing choreographies. In particular, there is no "framing" mechanism to delineate the constructs of choreographies from other choreographies' constructs and no naming mechanism. This task is to be implemented by tools. In consequence, choreographies essentially are identified by identifying start and end states that are interconnected via series of control flow states, PIPs, component choreographies and transitions. Figure 6.6 (a), (b) and (c) show different ways of representing a basic choreography that consists of two subsequent BT choreography tasks and does not contain any branching logic. Note that all three options basically express the same business information. In particular, it does not make a difference whether or not the choreography roles 'Customer' and 'Supplier' are explicitly mapped to PIP functional roles or not (confer Construct Advice 6.1.1). However, the benefit of separating choreography roles from PIP functional roles is evident. By means of choreography roles it is clear that the very same 'Supplier' role takes the 'Seller' role in PIP 3A19 and the 'Shipper' role of PIP 3B2 in figure 6.6.

As the BPMN standard does not constrain the labeling of choreographies, this thesis proposes to either assign the name of the choreography to its start state or to a text box placed in the "proximity" of the start state. If no visualization of the choreography's name is needed not visualizing the name is acceptable as well.

Construct Advice 6.1.6 (Representing Decisions)

Decisions are needed in choreographies to realize alternative control flow paths based on the result of BTs or choreographies. This thesis visualizes such decisions either implicitly by means of the set of outgoing transitions of a BT or component choreography or by means of BPMN *Exclusive Gateways* [150, section 11.6.1] that are represented as a diamond. Figure 6.7 (a) gives a sample for the former and figure 6.7 (b) gives a sample for the latter way of visualization. The condition expressions defined in "Construct Advice 6.1.4: Representing Transitions" are admissible for transitions. Modelers should use only one single incoming transition into BPMN exclusive gateways because condition expressions are defined relative to the preceding

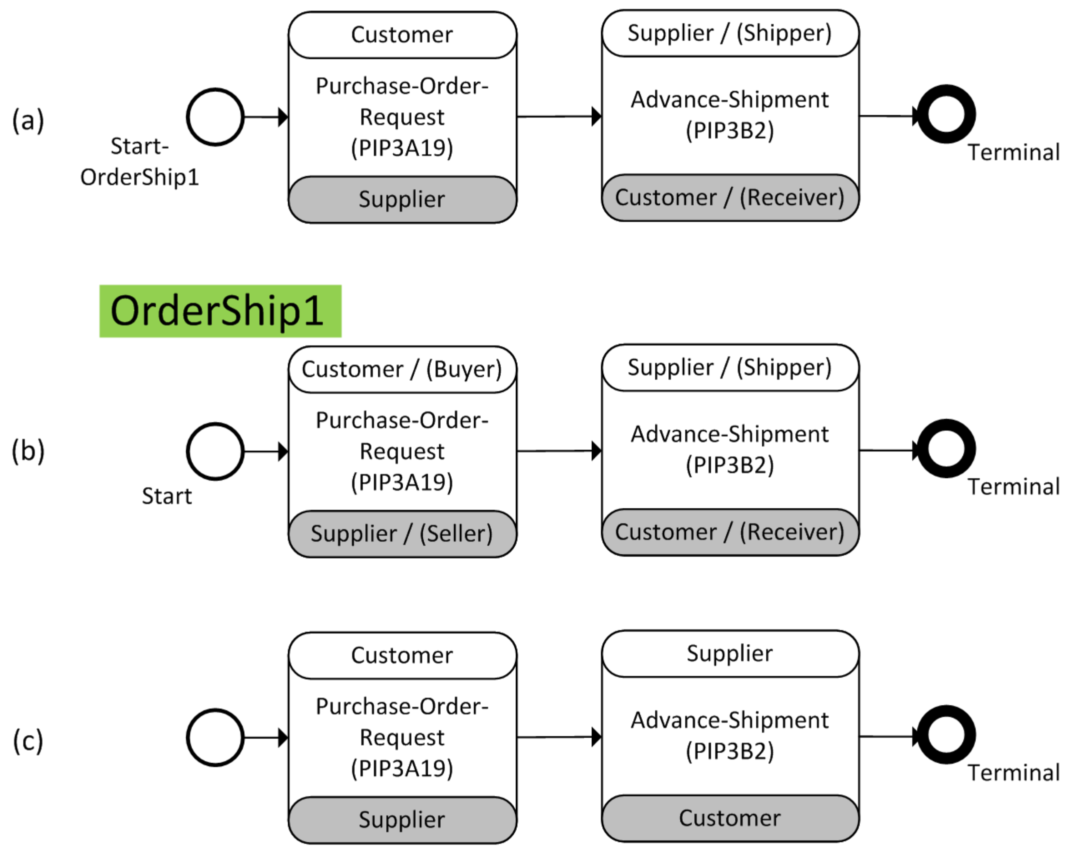


Figure 6.6.: Representing Choreographies

BT or component choreography. For example, XPath expressions are to be evaluated against the business documents exchanged during the latest BT instance.

It is good practice to define the condition expressions of the branches of a particular decision such that they are complete and disjoint. Completeness means that there is at least one condition expression that evaluates to true for any outcome of the preceding BT or component choreography. Disjointness means that there is no outcome of the preceding BT or component choreography for which more than one condition expression of the decision evaluates to true.

There is no evaluation order for the separate condition expressions of the branches of a decision because condition expressions have no side effects. The only exception is that CGV-based condition expressions are evaluated before any other condition expressions.

Exclusive gateways can but do not have to carry names. If names are assigned these should not contain spaces or special characters.

Construct Advice 6.1.7 (Representing Event-Based Choices)

An event-based choice is used within choreographies to realize alternative control flow paths based on events rather than based on the result of preceding activities.

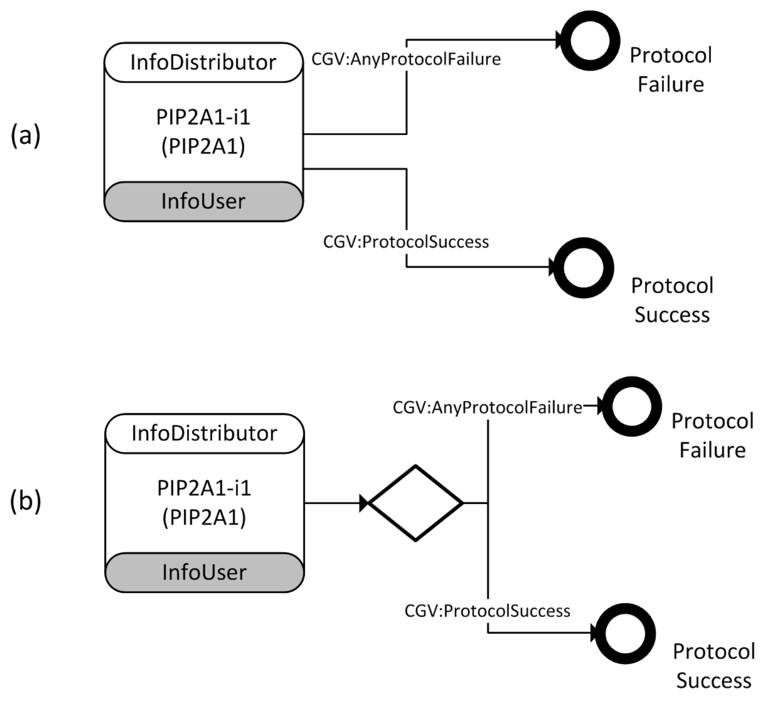


Figure 6.7.: Representing Decisions

The initiation of BTs and component choreographies or timeouts may be such events. BPMN *Event-Based Gateways* [150, section 11.6.2] that take the shape of a diamond with a double circle and a pentagon inside are used to represent event-based choices. Admissible events are timeouts, represented by BPMN timer symbols, and BT and component choreography initiations, represented by the corresponding visualizations. The semantics is such that the first event fired out of the events connected to the event-based choice triggers leaving the event-based choice. The implementation of this semantics is described in chapter 5. The BPMN standard makes restrictions about the types of events that can be combined after an event-based choice, for example, multiple follow-on choreography tasks are constrained to share a common requesting role. However, this work does not impose such restrictions upon modelers. Figure 6.8 demonstrates that it is perfectly acceptable to associate two BT choreography tasks with distinct initiators with the same event-based choice. Note that an event-based choice can be the target of one or more transitions as well as the source of one or more transitions.

Event-based choices can carry names. If names are assigned these must not contain special characters and must be unique relative to the enclosing choreography. Modelers may want to use this feature to document the progress of choreographies by assigning meaningful names (similar to the functionality of shared states). For example, the event-based choice of figure 6.8 carries the name ‘Accepted’ to highlight that a prior purchase order has been accepted. In this ‘Accepted’ state, either PIP 3B2 or 3A21 can be triggered or a timeout will be fired after 14 days.

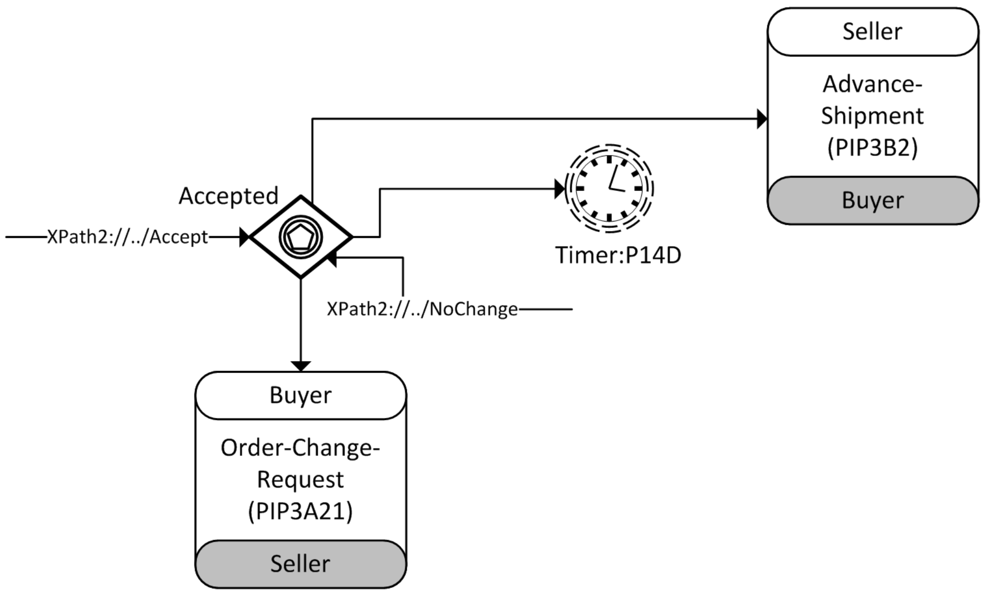


Figure 6.8.: Event-Based Choice Sample

Construct Advice 6.1.8 (Representing Parallel Structures)

Parallel structures may be used to define the possibility that two or more BTs or component choreographies can be performed at the same time. BPMN *Parallel Gateways* [150, section 11.6.4] visualized as diamonds with black, solid crosses inside are used to define parallelism. The semantics of this type of pseudo states depends on the number of transitions. It joins incoming flows if there is more than one incoming transition and it forks control flow if there is more than one outgoing transition or both. If a parallel pseudo state joins multiple incoming flows then no outgoing transition is fired until all preceding parallel activities have completed. Similarly, all outgoing transitions are fired and not only a subset thereof. In that sense, parallel pseudo states have "AND" semantics. It is good modeling practice to combine forking and joining gateways in pairs and not to define control dependencies between the individual branches. Note that it is considered to be acceptable to define a parallel pseudo state with only one single incoming and one single outgoing transition in this work.

Parallel gateways can but do not have to carry names. If names are assigned these must not contain special characters and must be unique relative to the enclosing choreography.

Construct Advice 6.1.9 (Representing Component Choreographies)

Component choreographies (BCAs in ebBP terminology) are used for encapsulating choreography logic and reusing it within larger parent choreographies. Using component choreographies, hierarchical decomposition of complex choreography models can be facilitated. BPMN offers two different options for visualizing component

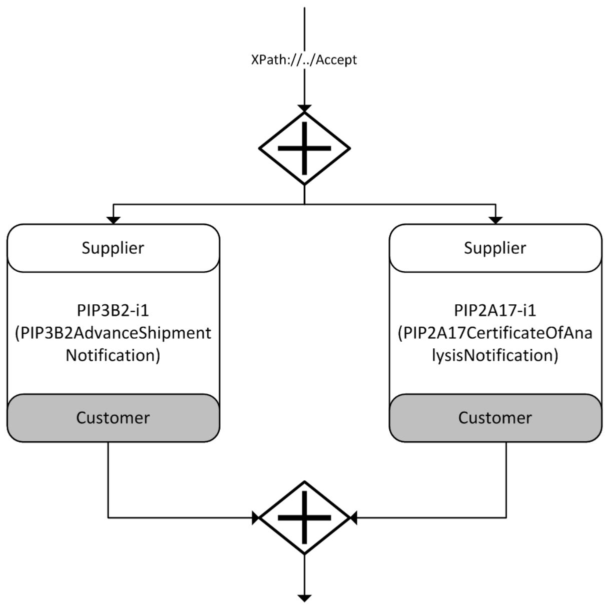


Figure 6.9.: Representing Parallel Structures

choreographies, so-called *sub-choreographies* [150, section 11.4.2] and so-called *call choreographies* [150, section 11.4.3]. While BPMN defines expanded and collapsed versions of both types only *expanded sub-choreographies* and *collapsed call choreographies* are used in this work for avoiding unnecessary complexity. Figure 6.10 and figure 6.11 both show the visualization of the component choreography named 'subchor' as an expanded sub-choreography. In these visualizations, the full definition of 'subchor' is given. Figure 6.12 shows the visualization of a component choreography as a call choreography task. The 'Invoicing' choreography of figure 6.11 is incorporated into the choreography of figure 6.12 by referring to its name 'Invoicing' within parentheses. While the full definition of the component choreography is available, it is not displayed within the parent choreography. The call choreography task itself carries a separate identifier 'c4' for being able to distinguish separate 'Invoicing' instances within the parent choreography (which are not displayed).

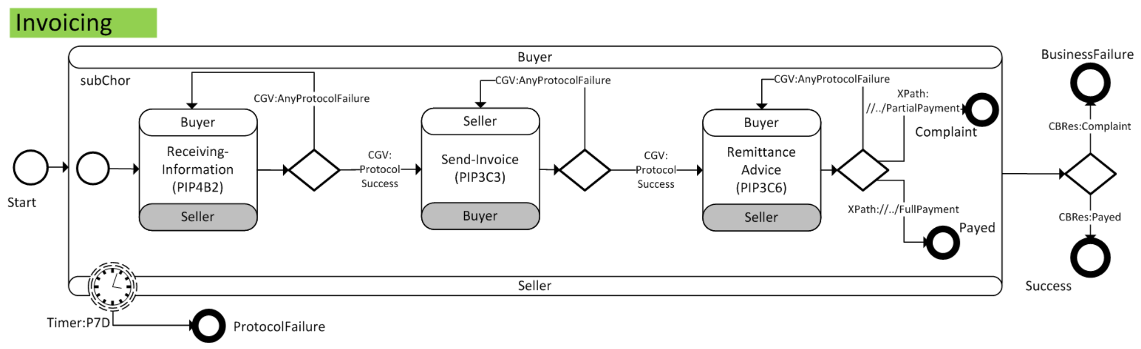


Figure 6.10.: Sample Expanded Sub-Choreography with Implicit Role Mapping

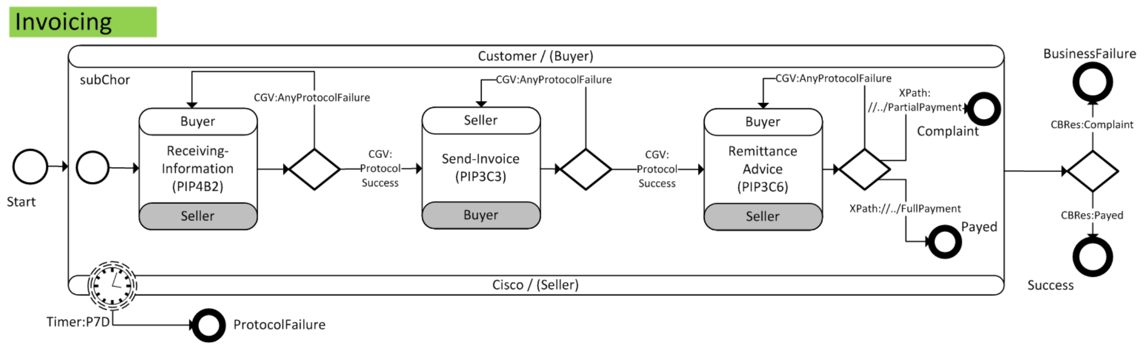


Figure 6.11.: Sample Expanded Sub-Choreography with Explicit Role Mapping

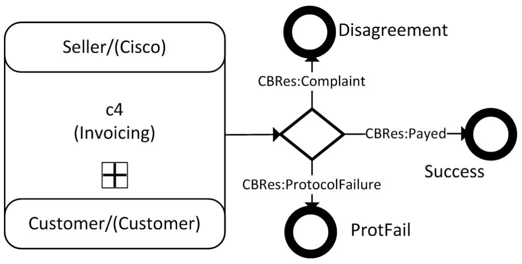


Figure 6.12.: Sample Call Choreography with Explicit Role Mapping

For component choreographies, the full set of constructs described in this section is available. However, some restrictions may apply depending on the choreography style (see next section). The semantics of component choreographies is such that when a transition to a sub-choreography or call choreography state is fired then the start state of the component choreography becomes activated and the parent choreography's state can only be left upon termination of the component choreography (except for timers, see Construct Advice 6.1.10).

In addition, the role definitions of the parent choreography must be mapped to the roles of the component choreography. Note that there always are distinct roles for the parent choreography and its component choreographies even if the same names are used. Therefore, in figure 6.10, there are 'Seller' and 'Buyer' roles at the parent 'Invoicing' choreography level and at the component choreography 'subchor' level. As no explicit role mapping is provided, the parent level roles are matched with the component level roles by means of string equality. The choreography of figure 6.11 semantically is identical to the one of figure 6.10 except for different parent level role names. Due to this difference the parent level roles are mapped explicitly to the component level roles using the notation introduced in Construct Advice 6.1.1. Explicit and implicit role mapping are available for call choreographies as well, although only the explicit version is exemplified in figure 6.12. Note that, from a technical perspective, the 'Cisco' role indeed is just a role definition that can be mapped to.

The results of component choreographies can be used to specify the control flow of parent choreographies by leveraging the CBRes expression language introduced in Construct Advice 6.1.4. Basic CBRes expressions refer to the names of the end states of component choreographies and become true when the corresponding end state is reached. More complex expressions can be created by using the standard Boolean operators to combine basic CBRes expressions. The use of CBRes expressions is exemplified in each of the figures 6.10, 6.11 and 6.12. At this point, it is noteworthy that figure 6.12 indeed has four choreography levels, namely the outermost choreography level depicted in figure 6.12, the ‘Invoicing’ and ‘subchor’ choreography levels of figure 6.11, and the choreography level for performing BTs according to the BT execution model of section 4.3.

For component choreographies with more than two roles, additional participant bands are added to the top or the bottom of the sub-choreography or call choreography shapes. In this case, it may be necessary to map one parent role to more than one component role. This is explicitly notated as follows:

ParentRole / (ComponentRole1, ComponentRole2, ...)

There are no constraints with respect to the sub-choreography or call choreography id names except for that special characters must not be used. In addition, those id names must be unique relative to the enclosing choreography. In case of global call choreographies, this means that its id name must be distinct from all other top-level choreography id names.

Construct Advice 6.1.10 (Representing Timeouts)

Timers come in two flavors, component choreography timers and event-based choice timers. These two types of timers reflect ebBP functionality and BPMN *interrupting* as well as *non-interrupting timer shapes* [150, table 10.90] are used to represent the corresponding events.

Component choreography timers are added to the boundary of sub-choreography or call choreography shapes and specify a time interval or a date and time that complies with the ISO 8601 standard. A component choreography timer has a follow-on node that is reached when a timer runs out. If the timer is interrupting (solid outer line of the shape) then the currently active task of the component choreography (if any) is interrupted and the timer’s follow-on node is immediately reached. If the timer is non-interrupting (dotted outer line of the shape) then the completion of the currently active task of the component choreography is waited for before the timer’s follow-on state is reached. Figure 6.10 above shows a non-interrupting component choreography timer that is attached to the ‘subchor’ sub-choreography shape and specifies a timeout after 7 days after the start of the sub-choreography (‘P7D’ is the ISO 8601 definition for Period-7-Days). Similarly, figure 6.13 shows an interrupting component choreography timer that is attached to call choreography task ‘c4’ and specifies a timeout after 12 hours (‘PT12H’ stands for Period-Time-12-Hours). Note that interrupting timers can be associated with sub-choreographies and non-interrupting timers with call choreographies as well.

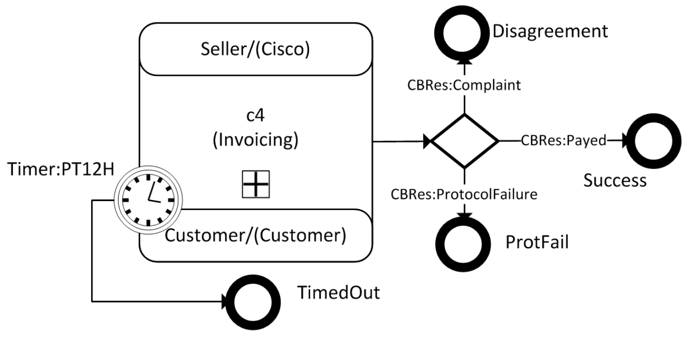


Figure 6.13.: Sample Scenario for an Interrupting Choreography Timer

Event-based choice timers (ebc timer) may be used as an alternative to choreography tasks after event-based choice states. ebc timers specify a time interval or a date and time and if the timer runs out before any of the alternative outgoing transitions of the respective event-based choice is fired then the follow-on state of the timer is reached. Figure 6.14 shows an ebc timer that is triggered if the ‘Buyer’ role does not initiate PIP 3A21 within 3 days.

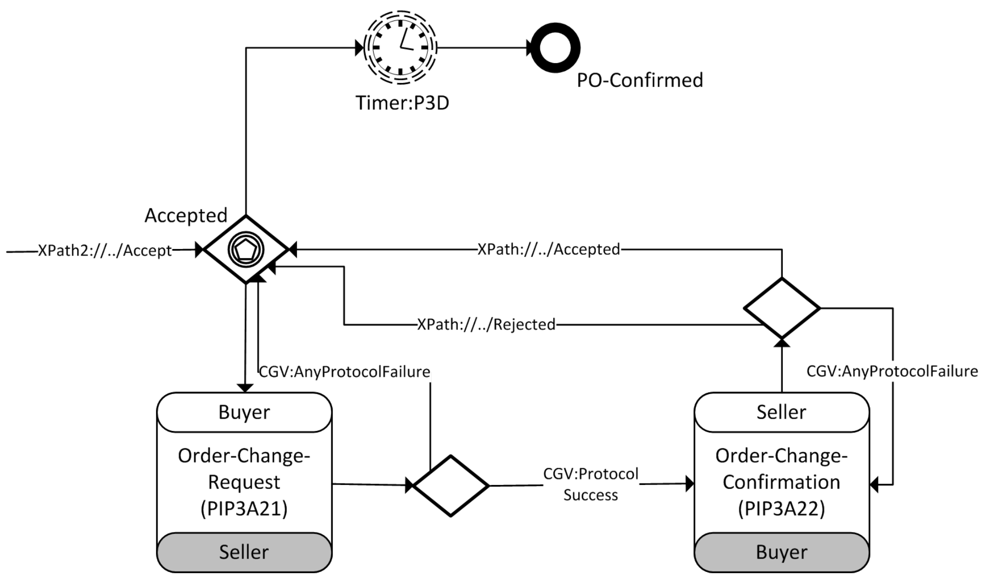


Figure 6.14.: Event-Based Choice Timer Scenario

In this scenario, the event-based choice ‘Accepted’ can be entered multiple times in case the Buyer initiates one or more PIP 3A21 executions. If such iterative behavior is part of a choreography definition it has to be decided whether or not an ebc timer is to be reset in case it is not reached for the first time. The default semantics is that an ebc timer is not reset. If reset is needed then a reset flag has to be added to the transition that enters the corresponding event-based choice. If an ebc timer runs out while some successor state of an alternative outgoing transition of the respective event-based choice is active then the event is not processed until the event-based choice is reached again. That means that ebc timers always are non-interrupting.

Note that ebc timers must not be confused with the *TimeToPerform* parameters of BTs. While ebc timers are controlled at the level of the surrounding choreography, the BT TimeToPerform parameters are controlled at the BT level.

It is required that labels of timers start with the string “Timer:” and then are followed by an ISO 8601 compliant definition of an absolute or relative date and time.

6.2. Representing Strict Choreographies

This section presents the rules for composing the BPMN elements introduced above such that resulting models correspond to the B2Bi choreography styles of chapter 4. Note that the *RosettaNet Methodology for Creating Choreographies* additionally specifies some basic rules for composing cartography style choreographies. However, these are not in the scope of this work. The presentation of model 'composition' rules is split up into section 6.2.1 for binary choreographies and section 6.2.2 for multi-party choreographies.

6.2.1. Strict Binary Choreographies

The following composition rules allow for choreography models that represent the superset of ebBP-ST and ebBP-Reg models. As ebBP-Reg represents the more expressive choreography style, the ebBP-Reg semantics of section 4.5.3 are the first choice for interpreting these models. Explicit modeling of shared states in the ebBP-ST spirit was discussed with the RosettaNet MCC team, but a completely new modeling element would have been necessary for that which was rejected for the sake of being able to use standard BPMN tools. Moreover, note that a strict binary choreography that uses event-based choices instead of shared states and refrains from using decomposition and concurrency can be interpreted as an ebBP-ST choreography from a control flow point of view. In consequence, the visualization still supports ebBP-ST models.

Note again that the following composition rules target a non-technical audience. For a formal characterization of valid B2Bi models, see chapter 4.

Strict Binary Chor. Rule 6.2.1 (Bilaterality)

Strict binary choreography models must define exactly two top-level choreography roles.

Strict Binary Chor. Rule 6.2.2 (Eligible Constructs)

Strict binary choreography models are restricted to the constructs described in the construct advices of section 6.1.

Strict Binary Chor. Rule 6.2.3 (Transition Coordination)

The processing of transitions is crucial for the execution semantics of process models. For strict binary choreography models, an outgoing transition of any node represents one option to continue the choreography. Two preconditions must be met for a transition to really fire.

Firstly, the transition must be enabled, i.e., the source of the transition must be a member of the set of the choreography's currently active states and if the transition carries a guard then this guard must evaluate to true.

Secondly, depending on the target of the transition, firing must be coordinated between the integration partners by means of requesting and confirming firing. Coordination ensures that only one out of multiple enabled transitions is fired. Furthermore, by means of coordination on starting BTs as well as component choreographies, both partners are aware of the fact that activities have been started. As a consequence, even protocol failures are valid activity outcomes to take routing decisions upon. Details on how such coordination may be implemented is described in section 5.3. The following list describes which transitions require coordination depending on the target state of the transitions:

- PIP/BT:
  Firing the transition must be coordinated.
- Component Choreography:
  Firing the transition must be coordinated.
- Fork state node:
  If the transition under consideration is the only enabled transition then firing may be coordinated. Otherwise, firing must be coordinated.
- Join state node:
  Firing the transition is performed immediately and must not be coordinated.
- Event-Based Choice:
  If the transition under consideration is the only enabled transition then firing may be coordinated. Otherwise, firing must be coordinated.
- Decision state:
  Firing the transition is performed immediately and must not be coordinated.
- End state:
  Firing the transition is performed immediately and must not be coordinated.

Strict Binary Chor. Rule 6.2.4 (Start States of Choreographies)

Strict binary choreography models must define exactly one start state as defined in construct advice 6.1.2. Figure 6.15 enumerates the different options, (a) to (e), for using start states to define the entry point into executable choreographies. Although not explicitly specified, the same options are available when defining a sub-choreography, i.e., in figure 6.15 (e) the three dots can be replaced with any of the listed options. Note that the start state of a strict binary choreography has exactly one outgoing transition that does not carry any guard and therefore immediately is enabled. Actual firing depends on the type of the target state as described in strict binary choreography rule 6.2.3.

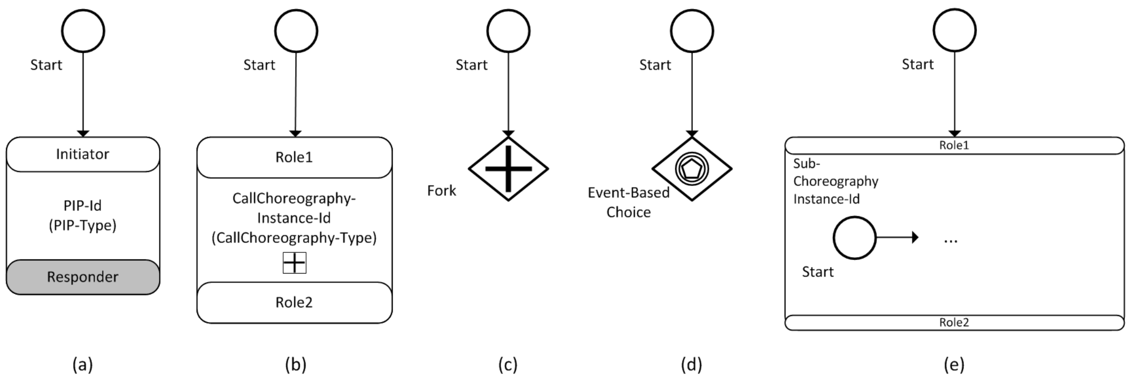


Figure 6.15.: Options for Starting Strict Binary Choreographies

Strict Binary Chor. Rule 6.2.5 (End States of Choreographies)

Strict binary choreography models must define one or more end states as defined in construct advice 6.1.3. Upon reaching an end state, strict binary choreographies terminate. In case the choreography under consideration is a sub-choreography of some superordinate choreography, the name of the end state immediately is propagated to the superordinate choreography and represents the result of the sub-choreography execution. The next state to be entered then is computed according to the semantics presented in strict binary choreography rule 6.2.7. If the choreography under consideration is a top-level choreography then no further activities are admissible. Note that due to the set of rules presented here, strict binary choreographies do not allow for multiple threads of the same choreography instance that terminate in separate top-level end states.

Strict Binary Chor. Rule 6.2.6 (BusinessTransaction Evaluation)

BTs are performed according to the execution model of section 4.3 and take time. A BT can only be left upon termination of the execution protocol which ensures that both integration partners have agreed upon the result of the BT execution, may it be a protocol failure or the contents of the exchanged business document. This result may be used for control flow routing purposes, i.e., to determine the follow-on state of the BT.

However, specifying unconditional progress without evaluating the outcome of a BT as depicted in figure 6.16 (c) is acceptable as well. Although that particular transition is immediately enabled upon completion of the BT, actual firing depends on the target state as described in strict binary choreography rule 6.2.3.

If the BT result is used for routing purposes then a set of ‘decision transitions’ is used to represent the branches of a decision that determines the follow-on state. Each of these decision transitions must carry a guard that complies with construct advice 6.1.4. Decision transitions are either added directly to the BT choreography task itself or to a dedicated decision state node. It is acceptable to combine both possibilities

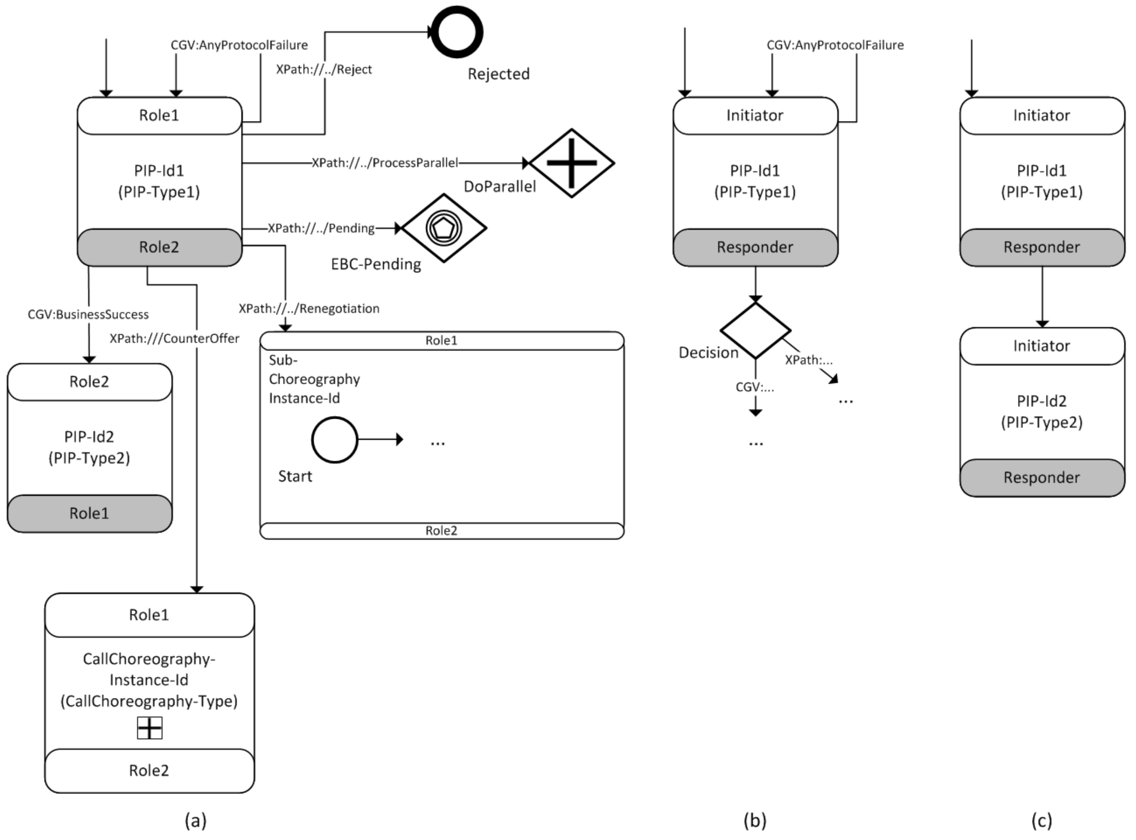


Figure 6.16.: Options for Continuing BusinessTransactions

for adding decision transitions (see figure 6.16 (a) and (b)). However, it is advisable to either exclusively add decision transitions directly to the BT choreography task or exclusively to a decision node. Note that a transition from a BT to a decision node does not carry any guard and is not in the set of decision transitions. Whether or not and when a decision transition is fired depends on the value of its guard and transition coordination as described in strict binary choreography rule 6.2.3.

Note that the BT configuration option *TimeToPerform* is controlled at the BT level and therefore not explicitly visualized using some timer construct at the choreography level. Therefore, timer constructs must not be added to BT choreography tasks.

Strict Binary Chor. Rule 6.2.7 (Component Choreography Evaluation)

Component choreographies either are call choreographies or sub-choreographies according to construct advice 6.1.9. The necessary steps for determining the follow-on states of component choreographies (evaluation) are identical for both types. Moreover, these steps are very similar to the rules for BTs (cf. strict binary choreography rule 6.2.6). Therefore, the considerations for unconditional progress and the optional use of dedicated decision nodes hold true correspondingly.

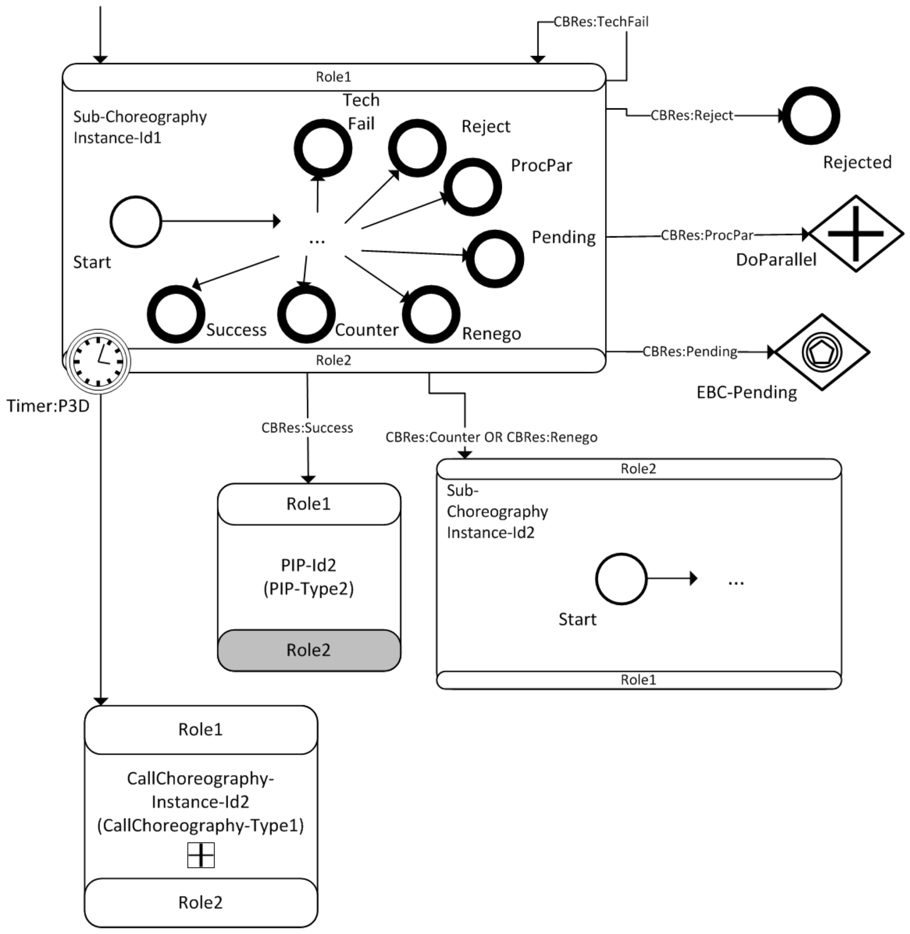


Figure 6.17.: Options for Continuing Component Choreographies

However, the guards of decision transitions are restricted to Boolean expressions built from the names of the end states of the component choreography (cf. construct advice 6.1.4). Figure 6.17 demonstrates how such guards can be used. Evaluation of the guards takes place upon component choreography termination. Again, the guard of a transition must evaluate to true for a particular transition to be enabled. Actual firing depends on the target of the transition and follows strict binary choreography rule 6.2.3. Component choreographies allow for the specification of interrupting and non-interrupting timers that are added to the boundary of the respective construct and that must comply with construct advice 6.1.10. In addition, timers have exactly one outgoing transition without a guard. A timer becomes enabled when a component choreography gets started and disabled when it terminates. If the timer runs out in between, the processing depends on whether the timer is interrupting or non-interrupting. In the former case, completion of the component choreography's currently active state is awaited for whereas the state is interrupted in the latter.

It is recommended that integration partners track the progress of the timed out component choreography in order to negotiate the business effect of timeouts that may go beyond determining the next state of choreographies. The following list discusses processing of timeout events for both cases for the respective types of states that may be the currently active state of a component choreography:

- Start state:
  The outgoing transition of the start state is immediately deactivated independent of whether the timer is interrupting or non-interrupting. This also holds true in case the integration partners already have begun coordinating upon the start of the start state's successor.
- BT:
  If the timer is non-interrupting, completion of the BT protocol is awaited for and processing of the component choreography is stopped thereafter. Then, the outgoing transition of the timer is activated. If the timer is interrupting, a cancel signal is sent to the BT execution protocol defined in section 4.3.2. If the cancel signal is accepted, the result of the BT execution is an AnyProtocolFailure. If the cancel signal is not accepted, which may be the case as the BT execution protocol may be in its finalization phase, then completion of the BT execution protocol's finalization phase is awaited for. Afterwards the outgoing transition of the timer is activated.
- Component choreography:
  This rule is applied recursively.
- Parallel structure:
  This rule is applied to each of the branches of the parallel structure.
- Event-Based Choice:
  All outgoing transitions of the event-based choice state are immediately deactivated independent of whether the timer is interrupting or non-interrupting.
- Decision state:
  Not applicable because the processing of decisions is assumed to take zero time.
- End state:
  Not applicable because the component choreography timers are deactivated upon reaching one of its end states.

If a timer runs out, the follow-on state of the choreography is determined by its outgoing transition. The point in time when the transition is fired complies with strict binary choreography rule 6.2.3.

Strict Binary Chor. Rule 6.2.8 (Event-Based Choice States)

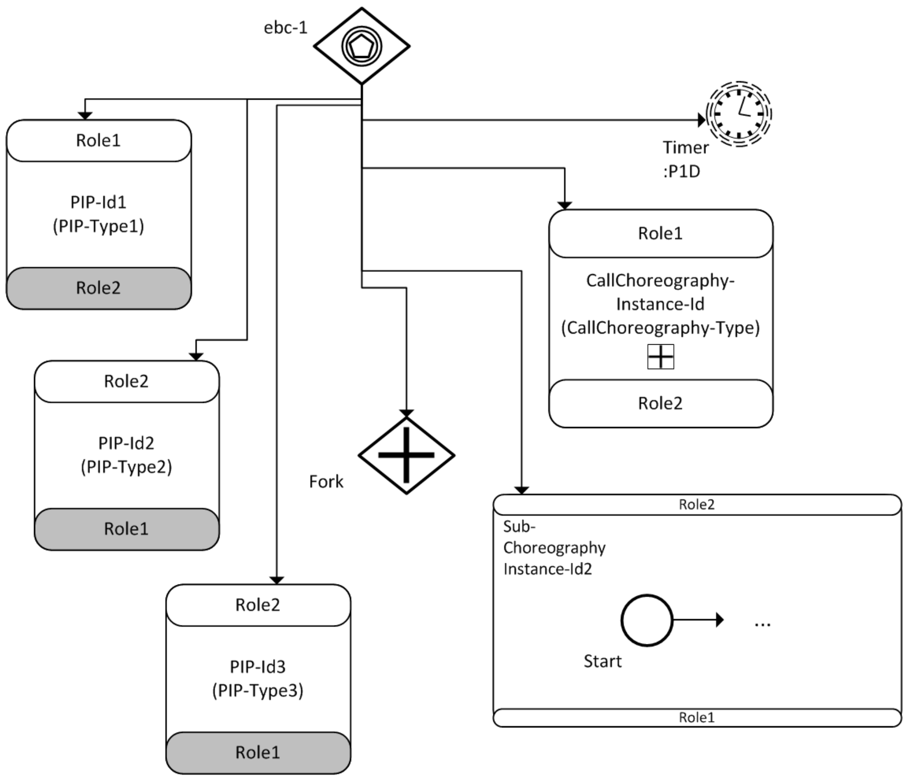


Figure 6.18.: Options for Continuing Event-Based Choices

Event-based choice states are used to select one out of multiple possible events. For strict binary choreography models, all outgoing transitions do not carry guards and therefore immediately are enabled. Except for timers, the integration partners are the source of events that trigger firing one of the transitions. As both integration partners may detect the need to take different transitions at the same time, they coordinate on which transition actually is taken. Therefore, specifying multiple BTs with different BT initiator roles as successors of an event-based choice is perfectly acceptable. One of the two integration partners takes the task of controlling a possible timer successor. If the timer runs out, this partner is responsible for coordinating with its partner that the event-based choice has been left by means of a timeout.

Strict Binary Chor. Rule 6.2.9 (Parallel Composition)

Parallel Structures in strict binary choreography models always are defined by means of a pair of fork and join states. One or more branches are defined between a fork state and a join state that are processed in parallel. Each branch is interpreted as a component choreography. However, this does not necessarily require the use of call choreography tasks or sub-choreographies as shown in the middle branch and the right-hand branch of the parallel structure of figure 6.19. If alternative constructs are used to start a branch of a parallel structure then the respective construct is interpreted as the first state after the start state of a virtual sub-choreography. So, for figure 6.19, the choreography task with instance id “PIP-Id1” is considered to be the initial state after a virtual component choreography’s start state. The grammar rules for creating choreographies then are applicable for creating the virtual sub-choreography. The only restriction of the rules is that exactly one transition to an end state of the virtual sub-choreography is replaced by a transition to the join state of the parallel structure. The virtual sub-choreography is assumed to be performed as if a dedicated sub-choreography construct was used. This means that multiple end states may be specified and that reaching any end state of the virtual sub-choreography results in reaching the join state of the parallel structure afterwards. For clarification, it is vital to note that figure 6.19 and figure 6.20 semantically are equivalent.

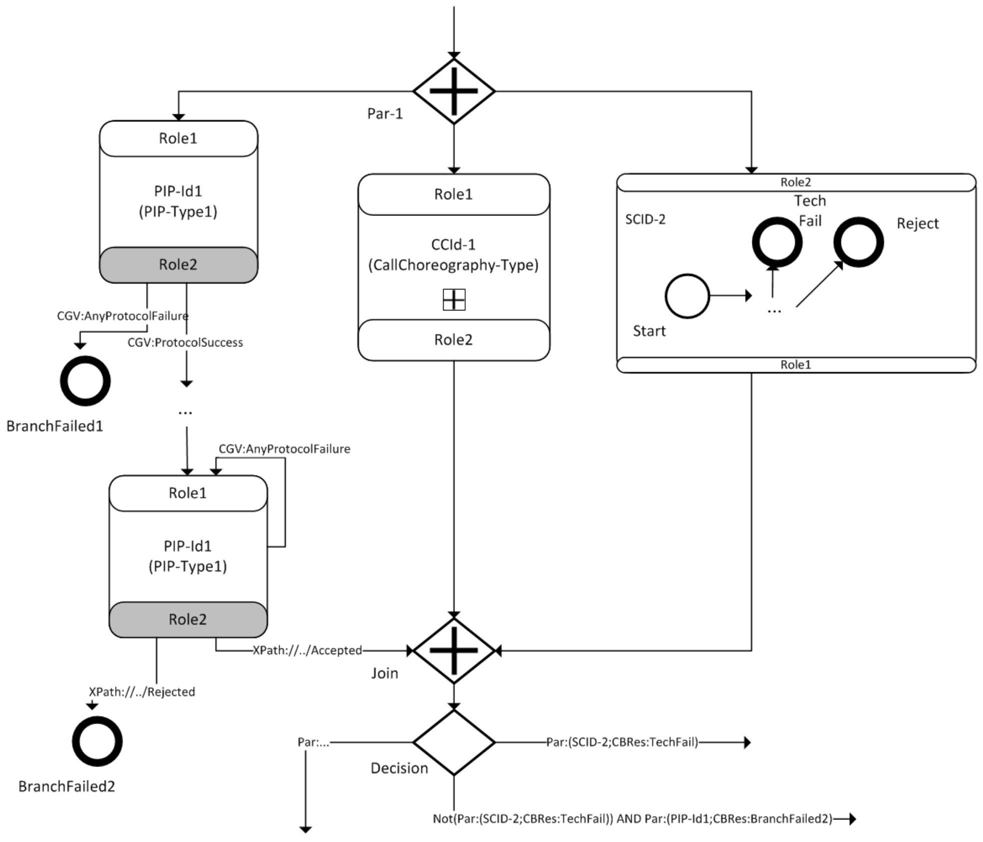


Figure 6.19.: Parallel Structure with Virtual Sub-Choreography

Although the use of virtual sub-choreographies for defining branches of a parallel structure may be convenient for modeling, the use of dedicated call choreography tasks or explicit sub-choreographies is advisable in order to avoid confusion about the interpretation of virtual sub-choreographies.

Note that the incoming and outgoing transitions to the component choreographies that make up the branches of a parallel structure do not carry any guards and immediately are fired. Moreover, transitions between the branches of a parallel structure must not be defined.

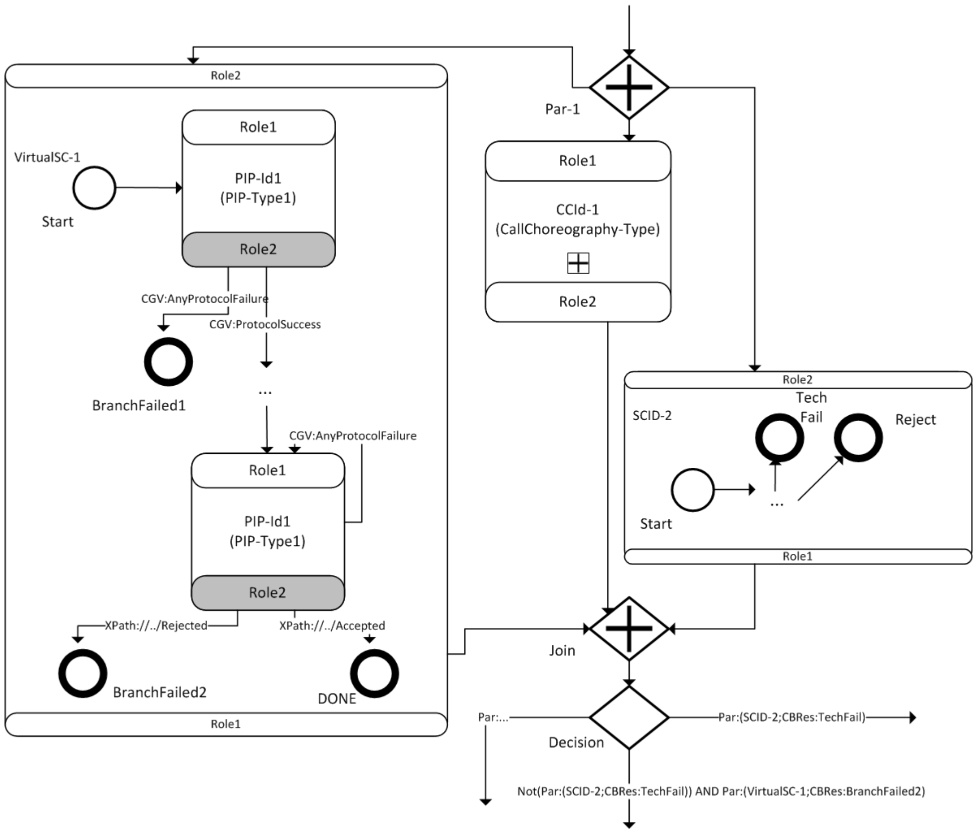


Figure 6.20.: Parallel Structure without Virtual Sub-Choreography

Finally, there may be the need for evaluating the results of a parallel structure's activities to determine the control flow after its join state. Therefore, the "Par" expression language is defined to be available for the outgoing transitions of a join state. Basically, "Par" expressions are built from the result expressions for the component choreographies that make up the branches of a parallel structure. So, basic Par expressions are 2-tuples, denoted as "Par:(Branch-Id;CBRes-Exp)", where Branch-Id is the id of such a component choreography and CBRes-Exp is a valid CBRes expression for the component choreography as described in construct advice 6.1.4. In case a virtual sub-choreography is to be evaluated, the Branch-Id refers to the id of the first construct of the sub-choreography. For example, the Par expression "Par:(PIP-Id1;CBRes:BranchFailed2)" is valid for the parallel structure depicted in figure 6.19. Moreover, the standard Boolean operators can be used to create complex Par expressions from basic Par expressions.

Strict Binary Chor. Rule 6.2.10 (Connectedness)

Not considering the direction of transitions, any state of a strict binary choreography model must be connected to the start state and all the end states of the same model.

Strict Binary Chor. Rule 6.2.11 (Guard Constraints)

For the set of decision transitions as used in strict binary choreography rules 6.2.6 and 6.2.7 the following must hold true:

- Completeness:
  The disjunction of the guards of all decision transitions must evaluate to true for any result of the evaluated activity.
- Disjointness:
  No two guards of decision transitions may both evaluate to true for the same result of the evaluated activity.
- BT ProtocolSuccess:
  If the evaluated activity is a BT then any non-CGV guard implicitly is AND connected with the expression "CGV:ProtocolSuccess".

Strict Binary Chor. Rule 6.2.12 (Producibility)

Strict binary choreography models must be producible according to the rules of this section.

6.2.2. Strict Multi-Party Choreographies

This section defines the composition rules for SeqMP choreographies as introduced in section 4.6. Remind that the purpose of SeqMP choreographies not only is defining multi-party choreographies (for an arbitrary large, but fixed set of roles) but in particular the analysis of synchronization deficits. The following rules that must be followed for defining valid SeqMP choreographies reflect this focus on analysis features:

SeqMP Rule 6.2.1 (Eligible Constructs)

SeqMP choreography models are restricted to start states, end states, transitions, decisions and component choreographies as described in section 6.1. The following rules abstract from whether component choreographies are represented as call choreographies or sub-choreographies. However, the component choreographies of SeqMP models must be valid strict binary choreography models as defined in section 6.2.1. Integration partners only may agree to use different types of component choreographies if they make sure that the end states of the respective component choreography is commonly reached by all component choreography roles. As a consequence, standard SeqMP models must not be used as component choreographies of other SeqMP choreographies.

SeqMP Rule 6.2.2 (Subsequent Role Participation)

In SeqMP choreography models, any two subsequent top-level component choreographies must share at least one top-level choreography role.

SeqMP Rule 6.2.3 (Transition Coordination)

Two preconditions must be met for a transition to really fire.

Firstly, the transition must be enabled, i.e., the source of the transition must be the SeqMP choreography's currently active state and if a guard is defined on the transition then this guard must evaluate to true.

Secondly, depending on the target of the transition, firing must be coordinated between the integration partners by means of requesting and confirming firing. Concrete coordination may be designed according to the definitions in section 5.3. The following list describes which transitions require coordination depending on the target state of the transitions:

- Component Choreography:
  Firing the transition must be coordinated.
- Decision State:
  Firing the transition is performed immediately and must not be coordinated.
- End State:
  Firing the transition is performed immediately and must not be coordinated.

SeqMP Rule 6.2.4 (Start States)

SeqMP choreography models must define exactly one start state as defined in construct advice 6.1.2. This start state has exactly one outgoing transition the target of which must be a component choreography.

SeqMP Rule 6.2.5 (End States)

SeqMP choreography models must define one or more end states as defined in construct advice 6.1.3. Upon reaching an end state, executable choreographies terminate. As SeqMP choreographies are not composable (see SeqMP rule 6.2.1), result propagation to higher-order choreographies is not applicable.

SeqMP Rule 6.2.6 (Component Choreography Evaluation)

The rules for determining the follow-on state of component choreographies in SeqMP models are very similar to the rules for determining the follow-on state of component choreographies in strict binary choreography models (cf. strict binary choreography rules 6.2.6 and 6.2.7). Therefore, the considerations for unconditional progress, for the use of the CBRes expression language and the optional use of dedicated decision nodes hold true correspondingly. However, decision transitions may only point to either component choreographies or end states.

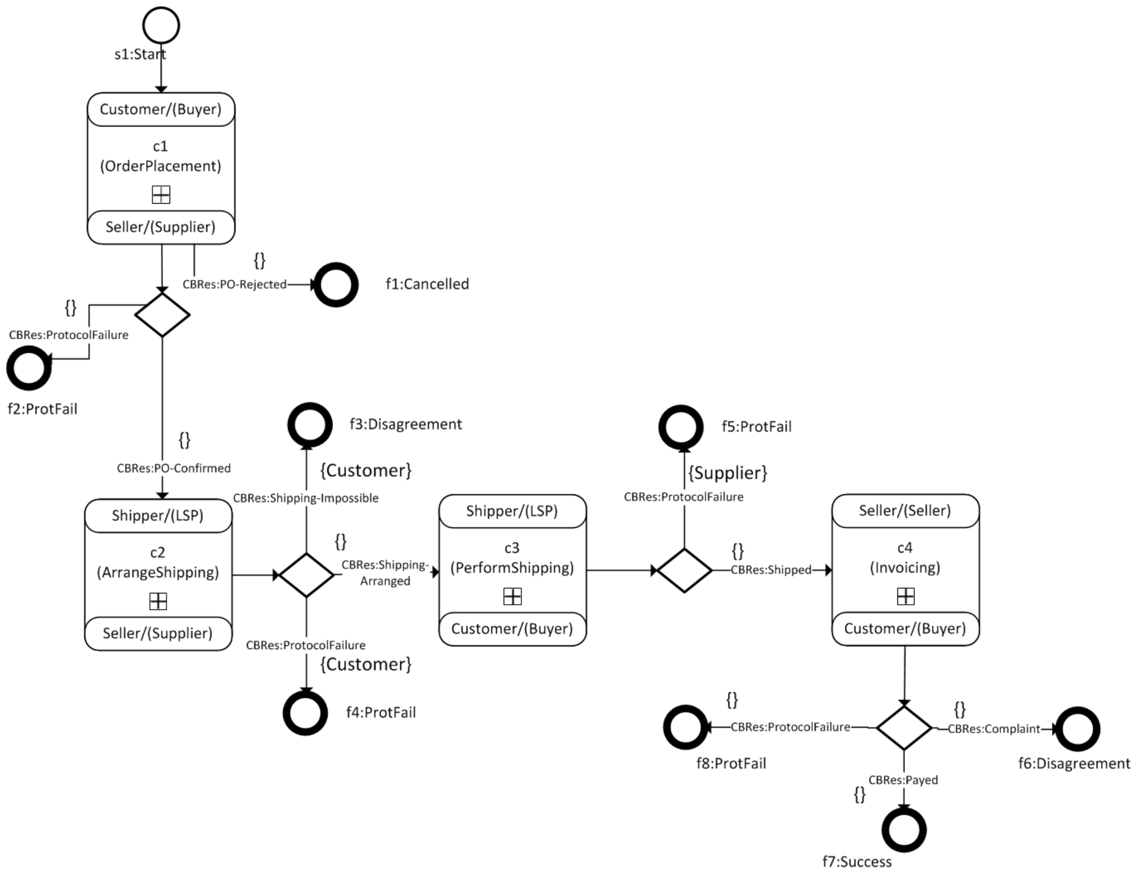


Figure 6.21.: Sample SeqMP Choreography

SeqMP Rule 6.2.7 (Escalation Assignment)

SeqMP choreographies are designed to analyze synchronization deficits that result from activities without participation of a particular top-level SeqMP choreography role. Consider the SeqMP choreography depicted in figure 6.21. The choreography starts out with an ‘OrderPlacement’ call choreography performed between the ‘Customer’ and the ‘Seller’ role. If the follow-on call choreography between the Seller and the Shipper fails, then the Customer is not automatically informed about that. However, the Customer may expect to participate once more in the overall SeqMP choreography, i.e., receive the product and invoice. Without explicit notification, the Customer may wait unnecessarily long which constitutes a synchronization deficit. SeqMP choreographies are not designed to avoid or automatically resolve such deficits. Instead, they provide a sound framework for identifying these. Therefore, so-called escalation sets may be added to transitions (using curly braces) to identify roles that may suffer from synchronization deficits upon firing the respective transition. Escalation sets are sets of top-level choreography roles and can intuitively be characterized as follows:

"If a role has already participated in the overall choreography and may participate in the future and is not participating in the current component choreography and if a transition is taken that excludes that particular role from further participation, then the role is to be included in the escalation set of that particular transition."

For example, figure 6.21 shows the escalation set 'Customer' for the decision transition after component choreography 'c2' with guard 'CBRes:ShippingImpossible'. This escalation set basically says that the Customer is the only one that may suffer (in the sense of having an information deficit) from the premature termination of the overall choreography upon firing this transition. The Seller and Supplier role do not have synchronization deficits because they participate in the source component choreography and therefore both have knowledge about the component choreography's result.

More details on the standard framework for analyzing information deficits, corresponding algorithms as well as precise rules for calculating escalation sets are described in section 4.6.

SeqMP Rule 6.2.8 (Connectedness)

Without considering the direction of transitions, any state of a SeqMP choreography model must be connected to the start state and all the end states of the same model.

SeqMP Rule 6.2.9 (Guard Constraints)

For the set of decision transitions as used in SeqMP rule 6.2.6 the following holds true:

- Completeness:
  The disjunction of the guards of all decision transitions must evaluate to true for any result of the evaluated component choreography.

- Disjointness:
  No two guards of decision transitions may both evaluate to true for the same result of the evaluated component choreography.

SeqMP Rule 6.2.10 (Producibility)

SeqMP choreography models must be producible according to the rules of this section.

6.3. Validation

The visualization of this work's B2Bi choreography styles serves two main purposes: Communicating B2Bi choreography construction guidelines to non-technical audiences and proving the amenability of this work to a model-driven software development approach that starts out with an abstract visual specification of the partner interactions. The discussion of validity is split up into two subsequent steps correspondingly.

Communication to non-technical audiences Concerning communication to non-technical audiences, the choice of BPMN choreographies as visual notation limits freedom of optimizing the visualization in terms of usability to a large extent. However, using an international standard such as BPMN in a B2Bi setting is more important than perfect specialized usability so that an empirical evaluation of the visualization is left out. Yet, acceptance of the visualization guidelines of this chapter by RosettaNet's MCC team indicates that a reasonable level of usability is given. In that regard, note that RosettaNet's MCC team included several non-technical members.

The contribution of this chapter is not choosing BPMN as choreography notation, but rather restricting BPMN choreographies to a set of constructs and grammar rules that is adequate for B2Bi. The methodological foundation for restricting BPMN is given by [229] who identify four main grammatical deficiencies of languages compared to the ontological domain these are applied to:

- *"Construct overload:*
  Several ontological constructs map to one grammatical construct.
- *Construct redundancy:*
  Several grammatical constructs map to one ontological construct.
- *Construct excess:*
  A grammatical construct might not map to any ontological construct.
- *Construct deficit:*
  An ontological construct might not map to any grammatical construct."

While BPMN choreographies provide the grammatical constructs for such an analysis, the B2Bi choreography styles of this work provide the ontological domain. Declaring the choreography styles of this work as ontological domain of B2Bi (at least from a control flow perspective) is backed by a comprehensive literature study on the one hand (cf. chapter 3) and an analysis of 100 *RIGs* of RosettaNet's RIG library on the other. *"[A RIG] describes the specific business scenario(s), usage notes and lessons learned [when implementing PIPs]"* which is supposed to *"help reduce implementation time and accelerate adoption of the process scenario by sharing* *the experience of early implementers."*[3] Control flow requirements of RIGs can be deduced from the business scenario descriptions and the overwhelming majority of RIGs is pretty simple in that regard. Only 44 out of 100 RIGs use hierarchical decomposition, 15 RIGs describe multi-party scenarios, 12 RIGs use loops, and 8 RIGs have parallel activities. As a consequence, the B2Bi choreography styles of this work cover the scenarios of the RIG library pretty well.

The visualization of B2Bi choreographies as presented in this chapter consists of just 10 construct advices, 12 rules for creating strict binary choreographies and 10 rules for creating SeqMP choreographies. Compared to the wealth of functionality available in the BPMN standard, this implies a considerable amount of construct excesses or construct redundancies in BPMN for the purpose of B2Bi choreography modeling. Nonetheless, some construct deficits were detected as well. Instead of discussing each BPMN element or modeling rule in isolation, a summary of deviations from the BPMN standard is given below. Note that restrictions and amendments to the BPMN choreography standard for the purpose of B2Bi choreography modeling are a natural thing because BPMN choreographies have not been designed as dedicated B2Bi choreography language.

The following BPMN extensions are defined:

1. Choreography tasks are interpreted as BTs that require the execution protocol defined in section 4.3 (cf. construct advice 6.1.1).
2. Expression languages for evaluating the result of BT executions are imported from ebBP (cf. construct advice 6.1.4).
3. Expression languages for capturing the result of component choreographies and parallel structures are defined (cf. construct advice 6.1.4 and strict binary choreography rule 6.2.9).
4. Basic rules for labeling top-level choreographies are defined (cf. construct advice 6.1.5).
5. A notation for role mapping from choreography roles to subordinate choreography roles is defined (cf. construct advices 6.1.1, 6.1.5 and 6.1.9).
6. A reset semantics for timers that may be entered from an event-based choice several times is defined (cf. construct advice 6.1.10).

The following BPMN rules are violated:

1. Guards are allowed to be added to transitions without a so-called "mini-diamond marker" which contradicts the following BPMN rule:
   "A conditional outgoing Sequence Flow from an Activity MUST be drawn with a

[3] `http://www.rosettanet.org/Support/ImplementingRosettaNetStandards/RosettaNetImplementationGuides/tabid/2985/Default.aspx`, last access: 12/20/2011

mini-diamond marker at the beginning of the connector" [150, section 8.3.13]. Not forcing the user to add "mini-diamond markers" to conditional guards follows the rationale that a guard *represents its own existence itself*, that means by attaching a guard to a transition it is already clear that the sequence flow is conditional. It is noteworthy that this does not exclude adding "mini-diamond markers" if intended by the user.

2. For an exclusive gateway, no evaluation order of conditions (except for that CGV expressions are evaluated first) is prescribed which contradicts the following BPMN rule:
   "In order to determine the outgoing Sequence Flows that receives the token, the conditions are evaluated in order" [150, section 13.3.2].
   As BPMN choreography diagrams are two-dimensional there is no obvious way for determining the execution order of multiple conditions with the same source. Moreover, the BPMN standard does not define how to determine such an order. Finally, the validity of the condition expressions of this thesis' B2Bi choreography styles do not rely on an execution order because they do not cause side effects.

3. There is no distinction between collapsed (expanded) sub-choreographies and collapsed (expanded) call choreographies by means of "line thickness" as defined in BPMN. Instead, only the shapes for collapsed call choreographies and expanded sub-choreographies are defined. There is only the concept of component choreography that unifies the concepts of call choreographies and sub-choreographies. The use case of just defining part of a larger choreography is supported by implicit role mapping whereas the use case of using a component choreography in several other choreographies is supported by explicit role mapping. However there is no need to define two different types of component choreographies for that.
   Therefore, the following visualization rules of BPMN are disregarded:
   "If the Call Choreography calls a Choreography, then there are two options:
   - The details of the called Choreography can be hidden and the shape will be the same as a collapsed Sub-Choreography, but the boundary of the shape MUST have a thick line (see Figure 11.25).
   - The details of the called Choreography can be shown and the shape will be the same as an expanded Sub-Choreography, but the boundary of the shape MUST have a thick line (see Figure 11.26)" [150, section 11.4.3]

4. Conditional expressions after join nodes (parallel gateways) are allowed for in order to capture the result of a parallel structure. This contradicts the following BPMN rule:
   "A source Gateway [of a conditional sequence flow] MUST NOT be of type Parallel or Event" [150, section 8.3.13].
   As the branches of parallel structures may produce results that are relevant for the subsequent activities of the respective choreography, evaluating these results

must be possible and hence conditional expressions after join nodes should be allowed for. In case the modeling tool in use disallows multiple outgoing transitions of a parallel gateway then a decision gateway can be interposed.

5. The follow-on BTs of an event-based choice may have different senders and receivers. For example, there could be two BTs A and B where role1 is assigned the requester role of A and the responder role of B whereas role2 is assigned the responder role of A and the requester role of B. This contradicts the following BPMN rule:
"On the right side of the [event-based] Gateway: either - the senders MUST to be the same; or - the receivers MUST to be the same." [150, section 11.6.2]
Choreographies in which two parties may concurrently trigger the next BT are common in practice and the execution of such structures is supported by CHORCH. Hence, these structures are allowed for.

It is vital to note that the above restrictions and modifications of the BPMN choreography standard are well-motivated by the choreography styles of this thesis which, in turn, are motivated by the requirements analysis of chapter 3. Moreover, no new shapes have been defined for the visualization of CHORCH's choreography styles. As only a few labeling rules for representing B2Bi domain concepts have been added to standard BPMN choreographies, modelers should be able to leverage existing BPMN tools for creating visual B2Bi choreography models. The amenability of such models to model-driven software development is investigated next.

Amenability to model-driven software development Concerning amenability of this work to a model-driven software development approach that starts out with an abstract visual specification of the partner interactions, the coverage of relevant control flow constructs, the derivation of lower-level artifacts, and the applicability of the formal execution semantics need to be investigated.

Concerning the coverage of relevant control flow constructs, 11 use cases taken from the RosettaNet RIG library as well as the RosettaNet "Order to Cash eBusiness Scenarios" have been modeled using the modeling guidelines introduced above. The use cases have been selected such that the different control flow constructs of the guidelines are covered. For demonstration purposes, some additional use cases have been selected that only cover very simple control flow. As this use case analysis was part of the development of the above modeling guidelines, all use cases could be modeled and the result is publicly available together with the corresponding ebBP models[4].

Those ebBP models were generated by hand with the following two purposes. Firstly, to find out whether or not ebBP *skeletons* can be derived automatically from the information available in the BPMN model and, secondly, to analyze the detail that

[4]`http://www.rosettanet.org/dnn_rose/DocumentLibrary/tabid/2979/DMXModule/624/Command/Core_Download/Method/attachment/Default.aspx?EntryId=9858`, last access: 12/20/2011

has to be added in order to create a complete ebBP specification from such a skeleton. As the above BPMN modeling guidelines are based on the B2Bi choreography styles of this work, ebBP skeletons were found to be derivable automatically for all use cases (by means of some simple *nameId* derivation schemes). Only some naming rules for modeling element labels had to be adjusted. As technological gap between ebBP skeletons and full ebBP specifications, the following details have been identified:

1. Selection of business signals:
   A BT can either have no business signals, a RA, an AA or both. This configuration has to be filled in depending on the information to be exchanged. The identification of a BT pattern as defined in ebBP may be helpful for that [134, section 3.4.9.1].

2. Selection of QoS characteristics:
   Ten QoS parameters such as reliability, authentication or integrity protection have to be filled in for completing a BT configuration (cf. section 2.3.1). Again, the identification of a BT pattern as defined in ebBP may be helpful for that [134, section 3.4.9.1].

3. BT TimeToPerform Value:
   Whereas timeouts for leaving event-based choices or for interrupting component choreographies are part of the choreography definition and therefore explicitly visualized, the maximum time to perform a BT is part of the BT execution model and therefore not visualized. A static value has to be defined or the *“TimeToPerform”* parameter must be configured to be negotiated at runtime.

4. Business document version:
   While a BT type exactly identifies the semantics of the business documents to be exchanged, there typically still are several versions of it in business document libraries like RosettaNet or OAGi. The exact version of the business documents to be used has to be filled in.

5. BT functional role names:
   The ebBP format requires the definition of a *“nameId”* for the requester and the responder role of a BT. For RosettaNet PIPs, the functional role name of requester and responder role can be looked up in the PIP directory and used as *nameId*. If such a mapping is not available, an auto-generated id can be used.

Note that the visualization of this work is aligned with the B2Bi choreography styles of this work and not with the complete ebBP specification. In that sense, the visualization of this chapter is not a true ebBP visualization as required by the standard itself:
“Any methodologies and/or metamodels used for the creation of ebBP definitions MUST at a minimum support the production of the elements and relationships contained in the XML representation of the ebBP technical specification and defined in the ebBP schema” [134, lines 603-605]. However, it is noteworthy that the

purpose of visualization is enabling users to model B2Bi choreographies and not ebBP documents.

Finally, the applicability of the semantics defined in chapter 4 is discussed. As the visualization rules for SeqMP are strictly aligned with the SeqMP definition, applicability is not an issue. However, strict binary choreographies are a superset of ebBP-ST and ebBP-Reg. There are strict binary choreographies that can be interpreted using the execution semantics of ebBP-ST while the majority of models can be interpreted using the execution semantics of ebBP-Reg. However, if the full set of control flow constructs of strict binary choreographies is used, the semantics of ebBP-Reg has to be slightly extended as follows:

- Condition expressions after parallel structures must be allowed for. Therefore the semantics rule *3: Leave XOR-Fork or AND-Join* on page 137 would have to be split up into two rules where the rule for leaving XOR-Forks could be retained unmodified and the rule of AND-Joins would have to evaluate condition expressions against the component R of an ebBP-Reg configuration that already keeps track of all the outcomes of BCAs (component choreographies).
- By analogy with the ebBP-ST semantics, timeouts processing must be added. The application of the ebBP-ST timeout processing style is easily possible because both, ebBP-ST and ebBP-Reg, share *state machines* as underlying paradigm.

In addition, virtual component choreographies as described in strict binary choreography rule 6.2.9 have to be converted into explicit component choreographies by means of a preprocessing step.

6.4. Chapter Summary

This chapter introduced the visualization of CHORCH's B2Bi choreography styles using the BPMN notation. Using BPMN as visual format is beneficial because it provides a notation that abstracts from the technical detail that is irrelevant for identifying the types and sequences of business document exchanges. Moreover, using a standardized notation has the promise of tool availability and accessibility to a large audience. Finally, the interaction-centric choreography paradigm of BPMN choreographies that is centered around choreography tasks lends itself well to visualizing the similarly interaction-centric choreography paradigm of ebBP that is centered around BTs.

The adaptation of BPMN choreographies to B2Bi choreography modeling takes the benefits of the CHORCH B2Bi choreography styles (cf. section 4.7) to an abstraction level that is closer to business process models:

- **Amendments of BPMN** As much as ebBP has been amended in chapter 4 in terms of redefining exchange procedures for BTs, the BPMN choreography notation is amended in adding B2Bi semantics and restricting the set of constructs used for choreography modeling. This enables B2Bi choreography modelers to reuse familiar concepts such as BTs on the one hand and reduces the number of modeling constructs and rules to bear in mind on the other. These two factors taken together promise a low barrier for adopting the proposed visual B2Bi choreography guidelines (cf. [126, 229]). It is worth noting that this thesis does not introduce any new visual constructs for modeling B2Bi choreographies. Only the semantics and labels of visual constructs are changed. As a consequence, the amendments made to BPMN choreographies do not challenge the availability of modeling tools.

- **Unambiguous execution semantics** Similar to ebBP, BPMN is a notation rather than a full-fledged modeling methodology. BPMN consistently does not define model classes such as ebBP-Reg, ebBP-ST or SeqMP that are aligned with specific modeling requirements (cf. chapter 3). However, BPMN defines the so-called *common executable conformance sub-class "that focuses on what is needed for executable process models"* [150, section 2.1.1]. This conformance sub-class prescribes the use of WSDL as service interface definition language [150, section 2.1.2]. However, WSDL belongs to a lower abstraction level than, for example, ebBP BTs the execution model of which is built on top of individual Web Services calls. As a consequence, an execution semantics for abstract B2Bi choreography models is not available in the BPMN standard.

  This thesis provides the BPMN world with such B2Bi choreography semantics by visualizing CHORCH's B2Bi choreography styles. The formal execution semantics of these styles can be reapplied to the visual choreography models defined in this chapter because the BPMN modeling constructs used can be mapped to ebBP modeling constructs one-by-one. Only technical configuration data has to be added for deriving complete ebBP models from visual BPMN models. Hence, a BPMN to ebBP transformation as proposed in this chapter does not challenge the defined control flow between BTs and BCs. In so far, the difference between the BPMN B2Bi choreographies introduced here and the ebBP choreographies of chapter 4 is more a matter of format conversion than model transformation.

- **Characterization of model validity** Consistent with the discussion in section 4.7 the term *"validity"* for the visual B2Bi choreography models of this chapter is tied to the ebBP-ST, ebBP-Reg and SeqMP choreography styles of chapter 4. As the BPMN models of this chapter correspond to the ebBP models of chapter 4, the validity criteria given in sections 4.4.3, 4.5.2 and 4.6.1 can be reapplied.

  However, these criteria are not very helpful for non-technical audiences. Therefore, the composition rules of section 6.2 present an informal characterization of model validity by defining admissible compositions of BPMN choreography constructs for the purpose of B2Bi choreography modeling. The figures contained in section 6.2 consistently strive for enumerating all valid successors of particular B2Bi choreography constructs.

The validity of this chapter's guidelines for creating visual B2Bi choreographies is given by relying on the B2Bi choreography styles of chapter 4. These, in turn, rely on the literature review described in chapter 3 and the analysis of 100 use cases taken from the RosettaNet RIG library. In order to check that the choreography styles of chapter 4 indeed can be modeled, 11 use cases that cover all relevant modeling elements have been modeled using the proposed BPMN choreography adaptation. Furthermore, these BPMN models have been mapped to ebBP by hand for ensuring that this translation step can be automated and for analyzing the semantic gap between BPMN models and ebBP models. Indeed, only technical configurations such as exact business document versions or security and reliability parameters have to be filled in for deriving ebBP models. This, in turn, implies that control flow structures that have been defined using the proposed BPMN visualization can be retained during the transition to B2Bi choreography implementations.

The visual notation presented in this chapter demonstrates that CHORCH's B2Bi choreography styles can be taken to an abstraction layer that is close to business process modeling. Conceptually, this is a step from the choreography layer as depicted in figure 1.3 of section 1.1.2 in an upward direction. On the contrary, chapter 5 shows how the binary CHORCH B2Bi choreography styles can be turned into implementations which is a step from the choreography layer towards the orchestration layer in a downward direction. In this sense, the contents of chapter 5 and this chapter validate the B2Bi choreography styles of chapter 4 by exemplifying the integration with surrounding technologies and abstraction layers. This shows that CHORCH's B2Bi choreography styles are amenable to software development approaches that begin with abstract visual models and then derive more and more concrete implementation artifacts.

The next chapter will discuss related work that is relevant for such approaches.

7. Related Work

The content of this chapter predominately covers analyses, concepts and approaches that compete with CHORCH or with particular concepts leveraged for CHORCH. In addition, some contributions that provide the background for CHORCH or complement CHORCH are discussed. The discussion of several alternative choreography languages and technologies contained in the previous chapters is not repeated here. This concerns the following topics:

- The description of several choreography languages that are not tailored to the needs of B2Bi as well as the discussion of UMM or BCL as alternative B2Bi choreography languages. Both aspects have been discussed in section 2.3.
- The reasons for not making use of additional ebXML standards, in particular ebMS and CPPA, the reasons for not leveraging REST, and the discussion of general purpose programming frameworks as alternative implementation platform. These aspects have been tackled in the introduction of chapter 5.

The discussion of related work starts out with some selected papers that provide additional background for this thesis.

In [14], Beimborn et al. give an overview of Web Services and ebXML standards relevant in 2002 and propose the integration of these two technology stacks. Beimborn et al. are among the first who suggest this integration. Furthermore, the discussion of [14] also implies the combination of high-level B2Bi specific description formats such as ebBP with general purpose implementation technologies such as Web Services. This backs up the levels of abstraction for representing choreographies and orchestration-based implementations as chosen by CHORCH. In so far, this work can be interpreted as a precise and detailed implementation of the high-level suggestions of [14].

Similarly, Khalaf et al. [73] suggest the use of BPEL and WS-Policy for implementing business processes which supports the choice of technologies in chapter 5.

In [249], Zapletal et al. give an overview of UMM version 2.0 and suggest the use of choreography languages that are tailored to B2Bi. Moreover, the need for incorporating a notion of state within B2Bi choreographies as well as the amenability of choreography models to automatic analysis and execution machinery is highlighted. This is very similar to CHORCH as it also follows the idea of applying a B2Bi-specific choreography format (ebBP). The state machine paradigm for CHORCH's B2Bi choreography styles as well as the concept of explicit shared-state nodes for ebBP-ST pay tribute to the importance of representing state in B2Bi choreographies. As CHORCH's B2Bi choreography styles are formally defined, have formalized execution semantics and can be serialized into ebBP, applicability of analysis and execution methods is guaranteed. It is worth noting that the content of [249] focuses on the language concepts of UMM and not on the methodology for applying UMM to concrete B2Bi scenarios. This is different from CHORCH that puts its emphasis on the application of existing B2Bi formats such that the semantics is unambiguous and executable. The authors of [249] have significant contributions in that domain as well which are discussed in section 7.2 and 7.3.

Further background for CHORCH is given by Naujok and Huemer in [121] by presenting an overview of the historic evolution of ebXML and corresponding successes and deficits. They find that the implementation features of ebXML have been outperformed by Web Services technologies and therefore achieved minor acceptance. To some extent, this is also true for the more abstract ebBP layer of ebXML. CHORCH dissociates ebBP from the ebXML stack and paves the way for applying Web Services technology to the execution of ebBP choreographies.

In that regard, Papazoglou discusses the relationship between Web Services technology and *business transactions* at length in [155]. A large part of the paper discusses the application of established transaction models such as short-lived ACID transactions and long-running business transactions. However, the details of leveraging these models for really performing business transactions are not defined. Instead, a high-level discussion of several deliverables of B2Bi communities such as RosettaNet is provided and a sample specification of a business transaction in a *"free XML syntax"* [155, figure 3] is given. That specification significantly resembles concepts of ebBP version 1 [147] (released two years before [155]) which is not fundamentally different from ebBP version 2 [134] in terms of the BT definition. In contrast to CHORCH, [155] does not provide a full-fledged execution model, execution semantics or a mapping to implementation artifacts for business transactions.

Another early discussion of B2Bi implementations is provided in [195] by Schulz and Orlowska. In particular, the separation of public integration logic from private integration logic and the use of process-based implementations is suggested. Although precise rules for creating process models or a definition of process model validity is not given, [195] captures important design drivers for B2Bi.

Another series of scientific approaches is concerned with choreography issues that typically are of minor concern for top-down approaches such as CHORCH. For example, Wombacher presents an approach in [235] for decentrally analyzing the consistency of multi-party interaction scenarios. Therefore, Wombacher captures the public behavior of communication partners as communicating state machines. Then, compatibility between public behavior definitions of communication partners is checked on a bilateral basis. A core contribution presented in [235] is the derivation of a criterion according to which checking the process compatibility between integration partners in a pairwise manner ensures overall compatibility of all integration partners in the sense that the cooperation between multiple partners may reach an end state. A tool for performing this analysis is presented by Wombacher et al. in [236]. Among others, the authors describe the derivation of the individual partners' state machines from BPEL processes. The contributions of [235, 236] are particularly useful for

analyzing the compatibility of potential existing integration partners without having to build a global multi-party choreography. This, in turn, implies a bottom-up approach rather than a top-down approach as applied by CHORCH. Moreover, B2Bi scenarios have been found to be mainly implemented on a bilateral basis (cf. chapter 3) so that the issue of creating a global multi-party model is of less concern for CHORCH anyway.
Several other approaches focus on the generation of adapter components in case a particular communication partner is found to be incompatible from a control flow point of view. In [71], Jung et al. present a rather informal approach for coupling the *executable processes* of two integration partners. To do so, one of the integration partners is suggested to externalize its publicly visible behavior as a so-called *contract process* and the other partner is supposed to adapt to this contract process. For bridging between executable processes and contract processes, the use of so-called *interface processes* is suggested. Although a state-machine like notation is used in [71] for visualizing sample processes, neither a formal definition of the various process types is given nor an algorithm for generating contract or interface processes. More formal in nature and sound in analysis are the adapter generation approaches presented in [241] leveraging communicating state machines and [21, 198] that are based on process execution trees. In a somewhat similar manner, the authors of [98] use Petri net technology for generating communication partners for sets of BPEL processes. Such approaches are of less concern for CHORCH because a global choreography definition is assumed to be defined collaboratively by the integration partners. Compatible processes using such a global choreography as input then can be generated as shown in chapter 5. This obviates the need for generating adapters or synthesizing communication partners. However, it is noteworthy that the concept of a process adapter bears some similarity with CHORCH's concept of *control processes* which underlines its validity from a software engineering point of view.

In order to round up the overview of approaches that *roughly* relate to CHORCH, it is worth noting that the choreography-orchestration dichotomy as coined by Chris Peltz [159] is mistakingly interpreted as choice of integration architecture from time to time. This means that some authors interpret choreography and orchestration as two distinct options for implementing the same task where choreography then is used to refer to the decentralized collaborative implementation of a common goal whereas orchestration is used to refer to the centrally controlled implementation. For example, McIlvenna et al. [107] present an approach for replacing the decentralized realization of an integration scenario with an equivalent realization that leverages a central process that serves as a kind of message hub between the interaction partners. The decentralized realization is declared to be a *choreography* whereas the centralized realization is declared as *orchestration*. In [7], Barker et al. compare the performance of a decentralized realization of a *Data-Intensive Computing* benchmark scenario to a centralized version. Again the decentralized realization is referred to as choreography whereas the centralized version is referred to as orchestration. There is a series of evidence that the concept of *choreography vs. orchestration as an integration architecture choice* is a misunderstanding. Chris Peltz states that *"executable [BPEL] processes model orchestration while abstract [BPEL] processes model the choreography of services"* [159]. However, as an abstract BPEL process does not collaboratively solve any task, this would not be a choreography according to [7, 107]. In addition, the concept of *local choreography* as used in [59] would not make any sense for the same reasons. Finally, following the concept of McIlvenna et al. and Barker et al. would allow for defining choreographies based on BPEL4Chor or BPMN collaborations that are orchestrations. All that needs to be done is building a BPEL4Chor model in which one BPEL process serves as message hub for the other BPEL4Chor participants.

The remainder of the related work discussion is aligned with the structure of this thesis. Section 7.1 presents related work for the requirements analysis (chapter 3). Section 7.2 discusses related work in terms of representing choreographies (chapters 4 and 6). Section 7.3 compares the implementation strategy of chapter 5 to alternative implementation approaches for choreographies and section 7.4 discusses approaches that target multiple abstraction layers of CHORCH.

7.1. Requirements Analysis for B2Bi

Related work for the requirements analysis of chapter 3 stems from the areas of Business Process Management, Supply Chain Management, Enterprise Application Integration, Information Systems Design, B2Bi and lots of more specialized domains that target specific integration technologies such as Web Services.

The discussion of that work is performed in two steps. At first, literature that targets the scope of the requirements analysis in full or partially is discussed. As pointed out in the underlying technical report [184], the scope of the requirements study used here is the *derivation of requirements for the analysis, design, development and maintenance of B2Bi information systems.* In a second step, the research areas that reside on the boundaries of this scope are identified.

Considering the first category of related work, the literature identified deviates from the analysis in chapter 3 with respect to scope or with respect to diversity of requirements sources or both. Basically, any of the requirements sources listed in tables A.1 and A.2 could be cited here. Instead, only those publications that come closest to the style of the requirements study are discussed.

Medjahed et al. discuss in [109] *issues and enabling technologies of Business-to-Business interactions.* Firstly, B2Bi interactions are split up into a *communication*, *content* and *business process layer* and, then, *coupling among partners*, *heterogeneity*, *autonomy*, *external manageability*, *adaptability*, *security* and *scalability* are identified as evaluation dimensions. Secondly, *B2Bi technologies*, *XML-based B2Bi frameworks* and *Web Services*, *Research Prototypes*, and *Deployment Platforms* are contrasted with these layers and evaluation dimensions. Finally, some open issues are specified. The work of Medjahed et al. is different from this study in offering a less elaborate B2Bi schema and in defining a less comprehensive set of B2Bi requirements where both the evaluation dimensions and open issues identified in [109] can be considered to define B2Bi requirements. Instead, the authors of that paper concentrate on a more detailed presentation of the technologies/frameworks/prototypes/platforms, but do not use these for gathering requirements.

In [242], Yu et al. study *issues, solutions and directions in deploying and managing Web Services*. That paper is related to the requirements study of chapter 3 in dealing with an important B2Bi implementation technology, but, as B2Bi is not the only application domain of Web Services, B2Bi specific characteristics are not treated comprehensively. Moreover, the focus of the paper is put more on the analysis of implementation concepts and technologies than on deriving requirements.

In [213], van der Aalst et al. survey business process management and therefore are relevant as well. They define the BPM lifecycle and put important related concepts like Workflow Management, Business Activity Monitoring and Business Process Analysis into context. In doing so, they also identify several important requirements for B2Bi, but they do not derive a comprehensive requirements list nor classify them according to a B2Bi schema.

In [3], Androutsellis-Theotokis et al. present a requirements analysis and design proposal for performing peer-to-peer e-business transactions. That paper focuses mainly on the *market B2Bi* type as identified in [48] and therefore is not as comprehensive in scope as the analysis discussed here. Moreover, the authors do not classify their requirements according to B2Bi challenges or abstraction layers.

In [176], Scheithauer and Wirtz present the results of a case study in business process management. This comprises a list of requirements that is less comprehensive than the one of this thesis. Moreover, only a single source for requirements, i.e., the case study, is considered for deriving requirements.

An Enterprise Integration (EI) Methodology is described by Lam and Shankararaman in [89] and they explicitly declare B2Bi to be a special EI scenario. The focus of that paper are the envisioned project phases of the integration methodology, but lists of *integration requirements* and *qualities of integration architectures* are also presented. Whereas the integration requirements only deal with issues related to technical communication like response time and volume/throughput, the qualities of integration architectures are boiled down to five abstract qualities such as openness and feasibility. Methodologically, that paper is different from the requirements analysis of this thesis in using Lam's and Shankararaman's experience in EI projects as requirements source instead of using different types of literature as done in [184].

Apart from the papers just discussed that have a more or less global B2Bi scope, there are lots of publications that are different from the requirements analysis of this thesis in focusing on selected aspects of B2Bi or BPM systems, e.g., process modeling languages [100], simulation [47], process flexibility [193] or semantic constraint management [101].

Finally, the B2Bi schema taken from [187] (see figure 1.3) is a source of requirements on its own. In particular the B2Bi schema layers are not explicitly declared to be a requirement, although it is obvious that there is a need for 'business process modeling' (BPM layer) or 'the definition of message exchange between partners' (public orchestration layer). These requirements are not included in the requirements set of tables A.1 and A.2 because they are part of the proposed B2Bi schema.

The second step in discussing related work of the requirements study consists of identifying research areas at the boundaries of its scope. It is vital to note here that the requirements study explicitly does not look at requirements of these areas.

The first area comprises *project management and organizational issues* during performing B2Bi projects. Project management typically comprises tasks like risk management, human resource allocation or project scheduling, while organizational issues cover aspects like cultural fit, implementing organizational change, level of support for IT projects or the analysis of organizational capabilities. Exemplary publications that are dedicated to these issues are [88] investigating EAI success factors or [119] defining a capability assessment framework for the adoption of B2Bi systems. This area is also considered to be important in research work with a different focus like [49, 90, 126].

Another neighboring area is the business perspective on B2Bi that drives the design and development of B2Bi information systems. Supply Chain Planning is a major part of the B2Bi business perspective and comprises long-term decisions such as the strategic design of an enterprise's supply chain network to mid- and short-term planning tasks such as collaborative planning, forecasting and replenishment (CPFR) or master planning. The definition, measurement and monitoring of performance figures also belongs to the business perspective of B2Bi. Clearly, all these tasks heavily influence the functional and non-functional requirements for a particular integration system, but the domain problems themselves first have to be solved using methods of logistics and supply chain management, corporate management and others. These domain problems are discussed, for example, in [163] that targets inter-domain master planning in supply chains, in [52] targeting performance measures for supply chains and in [175] dealing with the strategic design of supply networks.

Finally, as the scope of the requirements study comprises the design and development of information systems, generic requirements for models and implementations apply. Examples of such requirements are coupling, cohesion, abstraction or reuse. The requirements study for this thesis [184] does not attempt to define a comprehensive requirements list for arbitrary models. Instead, the goal is a comprehensive list of B2Bi requirements that can be justified by relevant B2Bi requirements sources.

7.2. B2Bi Choreography Representation

The related work in this section does not only cover approaches based on B2Bi choreography languages. It is interesting to note that virtually any choreography approach claims relevance for *interorganizational systems* or *B2Bi*. The main differences lie in the languages chosen for representing choreographies and the actual aim. While approaches based on languages with a strong formal background frequently strive for analyzing the soundness of choreographies, approaches based on standard languages rather strive for supporting the communication function of choreography models and streamlining implementation. The paragraphs below are structured according to the languages used for representing choreographies. Remember, though, that publications that predominately aim at introducing a new choreography language such as [248] (that introduces BCL) are covered in section 2.3. In addition, there are numerous publications that focus on the soundness of the service compositions of individual choreography participants. For example, the consistency between executable and abstract processes is investigated in [105] and a Petri net-based algebra for modeling Web Services compositions is presented in [53]. Such approaches are not further discussed here. Finally, approaches that target the derivation of implementation artifacts from choreographies are covered in the next section.

Petri Net-Based Choreography Approaches There is a significant number of choreography approaches that leverage Petri Nets[1] for modeling choreographies. These can be classified into interconnection choreography models and interaction choreography models (cf. section 2.3 and [29]).

In [214], van der Aalst and Weske present the *Public-to-Private approach* as one of the first interconnection choreography approaches based on Petri nets. The approach starts out with a global model in which places are used to either represent message buffers between interaction partners or the local state of an interaction partner. Transitions correspondingly represent receive and send actions that consume or produce tokens in message buffer places and local state places. The local state places and send/receive transitions of individual partners are arranged in (imaginary) lanes and thus represent participant specific behavior. The approach of [214] then provides guidelines for dissecting such a global model into the public participant behaviors and deriving *private* orchestration models of these public behaviors such that the overall defined choreography remains unchanged. Such a *local* Petri net model of a participant then can be used to derive implementations (not presented in [214]).

In [212], van der Aalst et al. provide a refined inheritance notion for deciding upon conformance of private orchestration models (expressed as Petri nets, too) to public participant behaviors. Dijkman and Dumas present an approach in [35] that allows for a more fine-grained modeling of public behavior. In addition to the global (choreography) model and the public participant behaviors of [214], Dijkman and Dumas define the concepts of *interface behaviors* and *provider behaviors*. An interface behavior describes the public behavior of a communication partner with respect to one particular other communication partner. In so far, the global choreography model of the Public-to-Private approach can be interpreted as a set of interface behaviors. A provider behavior as defined in [35] can be used to describe the public behavior of a communication partner with respect to all relevant other communication partners. In so far, a provider behavior in [35] is somewhat equivalent with the public participant behavior of [214]. Finally, Dijkman and Dumas envisage the implementation of a provider behavior by one or more *orchestrations* whereas [214] does not elaborate on this distinction.

[1]The reader is assumed to have a working understanding of Petri nets

Petri net-based choreography approaches such as [35, 212, 214] are different from CHORCH in several ways. Firstly, CHORCH provides interaction style choreographies whereas [35, 212, 214] provide interconnection style choreographies. Interaction style choreographies are said to suffer from the possibility of unenforceable models whereas interconnection style choreographies are said to suffer from the possibility of incompatible participant behaviors. Realizability of interaction choreographies can be ensured by a set of rather simple composition and execution rules as CHORCH shows. On the other hand, leveraging a formalism like Petri nets with its rich set of theory and analysis features allows for checking compatibility of interconnection choreographies. Hence, the choice between interaction and interconnection choreographies is rather dependent on modeling adequacy. A core motivation for choosing interaction style choreographies is the fact that all major B2Bi choreography notations, in particular UMM and ebBP, provide the interaction choreography style. Secondly, CHORCH provides B2Bi domain concepts such as BTs and business document version configurations which make it more appropriate for practical implementation. Thirdly, the abstraction level of CHORCH choreographies is higher than the abstraction level of these Petri net based approaches. CHORCH introduces additional control messages in order to provide exactly the control flow defined by the global choreography at runtime. This is different from these Petri net approaches that only allow for the *business messages* defined in the choreography model that are exchanged using asynchronous communication. Due to concurrent access to the communication channel and overtaking messages, certain communication scenarios hence are hard to realize. For example, assume that two communication partners should be allowed to send a business message at a particular point in the control flow that mutually exclude each other. Simply modeling two send transitions for both partners would not be enough because these send actions are local activities that cannot be synchronized without further communication. CHORCH abstracts from such rather system-level problems and hence does not force the modeler to think in message buffers and synchronization procedures. Fourthly, the bilateral choreography styles of CHORCH are tailored to a more specific integration scenario and therefore allow for automatically deriving fully executable participant behavior implementations based on BPEL. To the best of my knowledge there is no Petri net choreography approach such as [35, 212, 214] that allows for translating participant behaviors into fully executable process definitions based on a production level orchestration language. Instead, behavior stubs (see several publications below) are generated that have to be completed manually. Considering the extensional complexity of CHORCH's orchestration models, manual completion of such process stubs may be far from practically doable.

In [33], Decker and Weske present *Interaction Petri nets* as interaction style choreography language. In Interaction Petri nets, transitions represent a message exchange between exactly two interaction partners. Places are used to represent the pre-/postcondition of such message exchanges so that these places can be interpreted as capturing the progress of the interaction. Multi-party choreographies are represented by assigning different roles to different transitions. In addition, Decker and Weske present an algorithm for deriving projections for interaction partners and an algorithm for deciding upon *local enforceability.* In that regard, it is important to note that the global choreography model of [33] models transitions (messaging events) as if they were synchronous, but assumes asynchronous communication for the interaction partner projections which leads to the same issues as discussed for the interconnection-style Petri net choreographies above. The *local enforceability* notion characterizes global models that do not suffer from such problems. In so far, the above argument that CHORCH preserves the modeler from having to think in such technical dimensions by ensuring the modeled control flow by means of introducing additional control messages applies. Moreover, the algorithm for deriving participant projections is of interest (when comparing with SeqMP models) which will be discussed below. Furthermore, the arguments of lacking B2Bi domain concepts as well as the derivation of implementation skeletons of participant behaviors instead of fully executable processes (when compared to CHORCH bilateral choreographies) applies again. These arguments also apply to [27], where Decker and Barros present a similar choreography model called *iBPMN* based on BPMN 1.0. In particular, an informal mapping to Interaction Petri nets is defined for leveraging the participant projection algorithm. The concepts of *iBPMN* as choreography language itself have been superseded by the BPMN 2.0 standard. Alternatively, iBPMN could be said to have inspired choreography concepts of BPMN 2.0. As regards the suitability of BPMN as B2Bi choreography notation, please see chapter 6. Finally, Decker et al. point out in [28] again that the projection of global Interaction Petri nets to participant behaviors may result in misbehaving interactions due to asynchronous communication. In addition, they define a correctness criterion of Interaction Petri nets by relating these to *weakly terminating desynchronized nets* where the *desynchronization* of an Interaction Petri net results in a Petri net model that is similar to the global choreography model in [214]. Moreover, Decker et al. propose three strategies for resolving *misbehaving interactions.* These comprise defining priorities on messages so that conflicting messages are locally reordered upon receipt, temporally allowing for inconsistencies that then are resolved automatically, and negotiating about inconsistency resolution upon inconsistency detection. This is different from CHORCH that introduces additional control messages in order to avoid inconsistencies. In that regard, it is essential to note that these control messages are technical in nature and do not affect business semantics because they just ensure a globally consistent order of concurrent messaging events.

A further Petri net-based choreography approach is proposed by Lohmann and Wolf in [99]. The difference to the above Petri net approaches is that Lohmann and Wolf begin with the lifecycle of *artifacts* that are involved in a cross-organizational workflow and not with the workflow itself. Examples for such artifacts are *orders* or *debits* and Petri nets are used to define the lifecycle of these artifacts where the Petri net transitions correspond to operations on the artifacts. This is in line with approaches that formalize the behavior of objects or components such as [233,241] and then analyze the interactions between objects/components. Lohmann and Wolf focus on the composition of a global choreography from the artifact Petri nets. These global choreographies then can be used as contract between the integration partners (where [99] does not elaborate on the implementation). A global choreography as generated in [99] uses Petri net places to represent the preconditions and postconditions of operations on artifacts. Petri net transitions are correspondingly used to specify which integration partner is in charge of performing a particular operation on a particular artifact. The difference to approaches such as [35, 212, 214] with respect to the global choreography model is that the activities of one particular integration partner are not automatically arranged in (imaginary) lanes and that there is no distinction between message buffer places and local state places. When compared to CHORCH, the above arguments about providing B2Bi domain concepts, abstracting from low-level interaction semantics and automatically deriving fully executable participant implementations apply. However, it is noteworthy that Lohmann and Wolf consider the data perspective of interactions that is disregarded in CHORCH choreographies. Checking CHORCH choreographies for consistency with respect to the evolution of involved data objects is a desirable goal for future extensions.

State Machine-Based Choreography Approaches Another well-known formalism that is frequently used for representing choreographies are state machines, in particular communicating state machines. A clear distinction between interconnection and interaction style choreographies for state machine-based approaches is sometimes hard to make (cf. [24, 72, 108]). In such approaches, the messaging behavior of each interaction participant is represented as a separate communicating state machine. A tuple of such state machines then can be interpreted as an interconnection style choreography. In addition, a global state machine where each state represents an overall state of the choreography progress (in essence a tuple of participant state machines' states) may be synthesized from the participant state machines which would imply an interaction choreography. Therefore, the approaches below are not categorized according to the *interconnection-interaction* dichotomy. Moreover, it is worth noting that several approaches discussed in the introduction of this chapter are based on state machines as well [235, 236, 241].

In [15], Benatallah et al. represent choreographies as *protocol machines* that are composed from the state machines that represent the messaging behavior of individual participants. The fact that a transition in the global protocol machine corresponds to a send and a receive action in the local state machines implies the assumption of synchronous communication. Furthermore, the fact that messaging actions in the local models have polarity only, that means whether they represent a send or a receive action, but no role association implies that bilateral interactions are focused on. In so far, the choreography model represented by protocol machines as defined in [15] is similar to the choreography model of CHORCH's bilateral B2Bi choreography styles. However, the purpose of the approach of Benatallah et al. is different. They focus on issues such as compatibility and replaceability of interaction partners and not on the streamlined implementation of B2Bi choreographies. In addition, they begin with the local models of participants which corresponds to a bottom-up approach rather than a top-down approach. B2Bi domain concepts as provided by CHORCH are consistently missing. Moreover, as the composition of state machines is performed for the purpose of analyzing compatibility and replaceability, the abstraction level of these protocol machines is lower than ebBP-Reg's and ebBP-ST's abstraction level. In [15], no additional control messages are considered for enforcing the control flow defined by global protocol machines. Finally, no mapping to an implementation language is provided in [15]. Instead, the authors refer to [5] that contains a proposal for deriving BPEL implementations from such protocol machines. The BPEL generation strategy in [5] is limited in not providing fully executable BPEL processes (only stubs are created) and in mapping the state machine structure to a BPEL `flow` element with control `links` between the states. Hence, the mapping is limited to acyclic graphs.

In [72], Kazhamiakin and Pistore also provide a state machine-based choreography approach that targets analysis instead of implementation. They create a kind of state machine composition of multiple local state machine models in order to analyze the compliance of interactions to a choreography model aligned with the WS-CDL specification. As the purpose is analysis, the comparison to CHORCH's features is similar to the comparison between CHORCH and [15]. However, [72] explicitly aim at multi-party interactions.

The lack of B2Bi domain concepts, missing abstraction from low-level interaction issues as well as the derivation of implementation stubs (if any) instead of fully executable processes also distinguish the following approaches from CHORCH. However, it is noteworthy that all of these are aligned with analysis goals rather than with implementation of B2Bi scenarios. In [232], Wieczorek et al. introduce the *Message Choreography Modeling Language (MCM)* that represents bilateral interactions as a global state machine. In order to support testing of MCM models, the authors analyze the different views that interaction partners experience during the execution of MCM models in case asynchronous communication across FIFO or non-FIFO queues is applied. In [85], *local enforceability* and *absence of inconsumable messages* are analyzed for the same communication model. Similarly, Bultan et al. analyze the problem of *realizability* for a state machine based choreography model that allows for multiple interaction partners in [24]. In addition, they coin the notion of *synchronizability* that characterizes the model property that the set of possible message exchange sequences between several communicating machines does not change if asynchronous communication is used instead of synchronous communication. Finally, McNeile claims to investigate the problem of realizability for synchronous and asynchronous communication in [108]. Again, a global state machine model where the transitions represent message exchanges between partners is taken as input for such an analysis.

Additional Non-B2Bi Choreography Approaches The series of choreography modeling approaches in this paragraph does not use a formalism such as state machines or Petri nets as primary modeling formalism. Industry languages such as BPMN or proprietary languages such as Let's Dance [245] are used instead. It is interesting to note that this coincides with a different aim of these approaches which is more about streamlining the implementation of choreographies than analysis of message exchange sequences. However, none of the approaches in this section makes use of a dedicated B2Bi choreography language which results in a lack of B2Bi domain concepts. This particular fact will not be repeated when comparing the approaches of this paragraph with CHORCH.

At first, a series of interconnection-based approaches will be discussed.
In [22], Bruno makes use of UML activity diagrams for modeling the behavior of individual choreography participants and attaches send and receive events to the activities of the corresponding diagrams. Several activity diagrams taken together then make up an interconnection choreography. However, no notion of model validity nor formalized execution semantics as CHORCH provides are given. In addition, a mapping to BPEL is provided neither.
In [77], Kim models the message flow of ebBP BT patterns as interacting BPMN 1.0 collaborations. In so far, [77] competes with the BT execution model defined in section 4.3. However, Kim neither defines a formalized execution semantics, nor a validation based on model-checking, nor a mapping to BPEL as CHORCH does.
In [30], Decker et al. also model choreographies based on BPMN collaborations. A restricted set of these BPMN collaborations then can be mapped to BPEL4Chor. For the mapping of BPMN collaborations to the abstract BPEL processes that define BPEL4Chor participant behaviors, Decker et al. suggest to make use of the algorithm defined in [153]. The drawback of the translation strategy defined in [153] will be discussed below. An extended treatment of the topic is provided by (almost) the same authors in [32]. There, a detailed comparison to other choreography languages, among others ebBP, is given. Apart from subsuming ebBP as bilateral choreography format, which is wrong since the release of ebBP version 2 in 2006, the authors claim to support more service interaction patterns. Indeed, this judgment is correct. However, the requirements analysis of chapter 3 reveals that control flow for the type of B2Bi systems targeted by CHORCH is rather simple and therefore this can be considered to be a minor limitation. On the other hand, the more focused scope of CHORCH allows for deriving fully executable BPEL processes that implement the control flow of the respective partners whereas Decker et al. only provide abstract BPEL processes. Moreover, Decker et al. do not provide simple rules for creating valid models such as provided in chapter 6 nor do they define a formal execution semantics.
Hettel et al. [56] derive BPMN collaborations using the so-called *Semantic Object Model (SOM)* [41]. SOM is a methodology for systematically deriving the structural and behavioral views on business processes. A business process model as derived by SOM identifies the interacting roles, the dependencies between the activities of the role behaviors and the message exchange relationships between roles. A similar approach to [56] is presented in [164] where the emphasis on deriving BPMN collaborations according to the SOM methodology is put more on the meta models of the respective model representations. In that regard, it is interesting that Pütz and Sinz [164] subsume BPMN collaborations rather as workflow model language

than business process model language. Indeed, the number of low level specification and modeling constructs such as WSDL interfaces in BPMN's *common executable conformance sub-class* [150, section 2.1.1] suggests a lower abstraction level than typically applied for business process modeling. CHORCH does not address the problem of systematically deriving business process models, but assumes that the interaction partners already have a corresponding model (at least in their minds). In so far, the integration of CHORCH with the SOM methodology similar to [56,164] is an interesting area of future work. However, as CHORCH has a strong focus on the interactions between integration partners, the derivation of BPMN choreographies instead of BPMN collaborations is desirable.

In the next part of this paragraph, interaction style choreographies are discussed. In [244], Zaha et al. use Let's Dance as starting point for deriving participant behavior implementations based on BPEL. A special focus is put on *enforceability* of the overall choreography control flow by the interacting participant behaviors, in particular in the face of asynchronous communication. This is similar to several approaches above. As a consequence, the difference to CHORCH is that such problems are not circumvented by leveraging technical control messages as done by CHORCH. In addition, Zaha et al. define a mechanism for deriving BPEL skeletons only whereas CHORCH provides fully executable BPEL processes.
Finally, Bultan and Fu [23] use UML collaboration diagrams for specifying service conversations (which may be interpreted as choreographies). Again, they put the focus on *realizability* of such a global choreography model if executed using asynchronous communication. In so far, this approach again adopts a lower abstraction level for the purpose of analysis than CHORCH does for the purpose of abstracting implementation. In addition, Bultan and Fu do not provide a mapping to implementation artifacts at all.

B2Bi Choreography Approaches The discussion of approaches that target B2Bi choreography modeling/specification starts out with [78], in which Kim uses UML 1.x activity and scenario diagrams to specify the control flow of B2Bi choreographies. However, [78] provides neither a formalization of models, nor a notion of model validity, nor a formalized execution semantics, nor a mapping to implementation artifacts. On the other hand, Kim uses UML class diagrams for capturing the data modeling perspective of collaborations which is not addressed by CHORCH.
In [67], Ilger and Zapletal transform UMM revision 12 collaborations into ebBP 1.1 models. This underlines that the integration of UMM and ebBP is highly desirable, in particular for complex integration scenarios that require the alignment of data modeling and control flow perspectives. However, as pointed out in chapter 6, the representation of B2Bi choreographies in one single diagram was a major requirement for the MCC project. As a consequence, BPMN choreographies have been selected as visualization of CHORCH's choreography styles. Apart from that, [67] do not provide a formal model of B2Bi choreographies.
Huemer et al. show in [64] how UMM modeling can be supported by means of so-called *worksheets* that capture UMM stereotypes and tagged values (for details,

please see the UMM specification [208, 210]). These worksheets can then be used to generate UMM models. In addition, an XML dialect for representing the layout and content model of worksheets is proposed in order to support customizations. As regards BPEL code generation, Huemer et al. refer to [61] which will be discussed in the next section. However, model validity in terms of well-formedness rules as given in chapter 6, formal execution semantics or a BPEL mapping for BCs and not only BTs is not provided.

Structured Modeling Approaches This paragraph does not contain choreography approaches, but process modeling approaches that are of particular interest in terms of model validity. The authors of [26, 76, 165] advocate the use of *structured modeling* of workflow processes, that means process models in which control flow nodes of matching types occur in proper nestings. Structured modeling of processes is typically applied in order to avoid *misbehaving* models that contain deadlocks or multiple instances of the same activity. In addition, structured modeling allows for a rather straight-forward mapping to block-structured languages such as BPEL[2].
CHORCH does not require the modeler to specify choreographies in a block-structured manner. While this complicates the mapping to implementation artifacts, it allows for an almost unconstrained specification of control flow and hence promises a user-friendlier way of choreography specification. This pays tribute to the CHORCH principle of valuing user requirements higher than infrastructure concerns.

Related Work for SeqMP's Information Deficit Analysis and Role Projection Algorithm The previous paragraphs of this section focused on comparing approaches to CHORCH's bilateral choreography styles. SeqMP choreographies are different because they are conceived as analysis framework and not as implementation contracts. In what follows, approaches that are relevant for SeqMP's information deficit analysis and role projection algorithm are analyzed. This comprises selected approaches already discussed above.

Generally speaking, SeqMP choreographies belong to the domain of business process management. In this domain, research on developing implementations or analyzing artifacts at the implementation level as presented in [84, 91, 168] is very common. However, SeqMP is substantially different. SeqMP choreographies provide a framework for analyzing multi-party choreographies at an abstract level. This study hence neither strives for automatically solving the partial termination problem nor for deriving implementations of the choreographies or of choreography projections.
The problems identified in section 4.6.2 are quite different from several problems identified in related choreography research. In reports such as [24,28,85,244], problems like *enforceability* or *realizability* are researched. The corresponding approaches have in common that the atomic building blocks of choreographies are single message exchanges and it is then researched whether or not the message sequences in the choreographies can be *enforced* by the local role-specific projections of interaction

[2]BPEL is predominately block-structured because only acyclic control flow graphs are allowed for.

partners, and whether or not the sequence of message exchanges is the same for synchronous or asynchronous communication. For SeqMP choreographies, these problems are not relevant. By using BCAs instead of single message exchanges, it is ensured that the state of integration partners is aligned at the end of each BCA (cf. [160, 190]). BCAs are performed using corresponding protocol machines that ensure alignment and therefore are not comparable to single messages in some communication buffer that may cause diverging states or deadlocks. Moreover, due to the *Subsequent role participation* condition, valid SeqMPs do not suffer from a local enforceability problem. To the best of my knowledge, *partial termination* as defined in section 4.6.2 has first been identified as multi-party choreography problem in [189].

However, deriving role projections of multi-party interactions has been a research topic for a long time. For example, the Public-to-Private approach of van der Aalst and Weske [214] (cf. above) dissects Petri Nets that contain role specific behavior. However, communication between participants is modeled as *send*- and *receive*-transitions that already are associated with roles. This corresponds to dissecting interconnection style choreographies and is representative for other interconnection style choreographies. However, SeqMP choreographies raise the problem of deriving role projections from interaction style choreographies. At first sight, the proposal for deriving role projections of *Let's Dance* models as described in [244] is comparable to that problem. However, Let's Dance applies a block-structured approach for modeling loops which is different from the class of SeqMP models with arbitrary loops. In [33], so-called Interaction Petri nets that enable multi-party interactions by modeling binary message exchanges as transitions of Petri nets, are analyzed. The model of [33] therefore can be considered to be comparable for the role projection problem of the work at hand and an algorithm for calculating role projections is presented as well. However, the solution in [33] is defined for Petri Nets and therefore the algorithm is not directly amenable to SeqMP models. Furthermore, the algorithm in [33] introduces duplicate transitions for flattening parallelism. However, transitions correspond to message exchanges in [33] and to BCAs for SeqMP. The business semantics of duplicate message exchanges (or duplicate BCAs for SeqMP) is not clear, though.

Apart from that, interesting choreography work has been presented in [59] where so-called local choreographies are used to model the sequence of interactions of one integration partner in multiple global choreographies. The perspective is different from the work at hand by focusing on a partner that participates in more than one choreography. The participants of the respective choreographies, in turn, may only know the focal integration partner. A typical scenario for that type of integration is a manufacturer that employs a sub-contractor producing parts of a product without the customer knowing the sub-contractor. While there may be valid reasons for not revealing the interactions with a particular business partner to different business partners, the type of integration in [59] leaves out the opportunity to perform analyses like escalation set computation that rely on a global view on choreographies.

7.3. Implementation of B2Bi Choreographies

In this section, the emphasis is put on related work that makes use of interacting processes for implementing choreography descriptions. This is the approach of CHORCH, too. In that regard, the most important category is the translation of choreographies into orchestrations. As CHORCH's choreography styles are graph-structured and BPEL is predominately block-structured, the first paragraph is dedicated to the translation of graph based control flow definitions into block structured definitions. Thereafter, approaches concerned with deriving orchestrations from non-B2Bi choreographies and B2Bi choreographies, respectively, are discussed. Finally, selected approaches that leverage significantly different execution concepts are compared to CHORCH.

Translation of Graph Structures into Block Structures In the last section, structured modeling of process definitions is identified as one way to smooth out the derivation of implementation artifacts from these process definitions. The basic idea is that matching control flow nodes of graphs are arranged in proper nestings so that each pair of control flow nodes delineates a block of control flow. If properly nested, the delineating control flow nodes are the only way to enter or leave the respective block. Hence, such blocks can be folded or represented by the constructs of a block-structured language. For a more detailed overview on block-structured modeling, see [76]. In [97], Liu and Kumar identify a series of unstructured process types (or graphs) that can be (or not) converted into structured ones. However, it turns out that not all unstructured graphs can be transformed into structured graphs. In particular, graphs that contain irreducible loops cannot be turned into block-structured languages by just folding control flow blocks (cf. section 2.2). In [111], Mendling et al. try to identify a taxonomy of translation concepts for turning graphs into block-structured language representations and vice versa. However, the strategy applied for translating ebBP-Reg into BPEL (cf. section 5.3) is not identified in [111]. The ebBP-Reg translation strategy emulates a state machine at the BPEL process level by means of a global while loop in which the current state is selected using a switch (which will be called the *state machine controller strategy* from now on) on the one hand and combines this with a threading-like execution strategy for the purpose of process decomposition. In addition, the taxonomy of Mendling et al. is incomplete in leaving out the translation strategy of Hauser and Koehler which will be presented next.

Hauser and Koehler present in [55, 80] an approach for automatically deriving BPEL code from arbitrary graphs that is based on a goto elimination algorithm provided by Ammarguellat [2]. Hauser and Koehler also identify the state machine controller strategy, but refuse to use it for two reasons. On the one hand, Hauser and Koehler are concerned about the performance impact implied by having to check the current node in each iteration of the global while loop used for implementing the state machine controller strategy. This is of minor concern for CHORCH because

B2Bi choreographies typically are not very large and state switches typically occur every few hours (when BTs terminate). In such a scenario, the linear time complexity of having to check n/2 guards on average per state machine controller iteration is negligible. In addition, the computing power consumed for checking a simple state variable typically is orders of magnitude lower than the computing power consumed for serializing or deserializing XML messages that are exchanged by business service interfaces. So, checking the current state of a choreography is *not* the bottleneck of B2Bi choreography implementations. The second reason based on which Hauser and Koehler reject the state machine controller strategy is that process structure is not visible in BPEL processes. However, the comprehensibility of fully executable BPEL processes is pretty limited anyway so that adequate visualization strategies have to be applied. Moreover, the goto elimination method proposed by Hauser and Koehler encodes part of the control flow in boolean flags which may become very complicated if irreducible loops are supposed to be translated. In so far, the readability of these BPEL representations is limited as well.

In [250], Zhao (and Hauser) et al. combine the mentioned goto elimination algorithm and the state machine controller strategy. For irreducible parts of a graph the state machine controller strategy is used whereas the goto elimination algorithm is used for the remaining graph. It is vital to note that the work presented in [55, 80, 250] is not about translating a choreography into interacting processes, but about turning a control flow description into its corresponding implementation. In so far, a BT execution protocol or coordination protocols for synchronizing control flow as offered by CHORCH are not in the scope of these contributions. Moreover, the rather general setting of these approaches does not allow for deriving fully executable BPEL process definitions as CHORCH does. These arguments also hold true for the remaining approaches of this paragraph and therefore will not be repeated.

In [151], Ouyang et al. propose the translation of arbitrary graphs into BPEL based on so-called *local partnerLinks*. A BPEL process sends a message to itself via such a local partnerLink. Then, defining BPEL event handlers that pass on control flow among each other by means of sending messages via local partnerLinks allows for encoding arbitrary graphs. The drawback of this approach is that local partnerLinks are not supported by all contemporary BPEL engines. Moreover, the strategy of Ouyang et al. requires sending a SOAP message from the BPEL process to itself every time a transition of the underlying graph has to be fired. The performance impact of passing a message through the Web Services stack is typically significantly higher than checking a global state variable as required for the state machine controller strategy. So, if the performance concern of Hauser and Koehler [55, 80] is of any relevance then the strategy in [151] is not a viable way for deriving BPEL implementations of graph-based process definitions. Finally, a local partnerLink as defined in [151] does not comply with the BPEL standard because it is used for incoming and outgoing BPEL messaging activities alike. This is not admissible because the local partnerLink definition in [151] uses the `myRole` attribute to identify the `portType` in the corresponding WSDL file. This should be done via the `partnerRole` attribute for outgoing messaging activities according to the BPEL standard. However, this problem can easily be circumvented by defining two partnerLinks.

Another strategy for translating graph-based process structures is trying to identify foldable blocks within graphs. Lassen and van der Aalst [92] use this strategy to translate Workflow nets (a Petri net dialect) into BPEL and try to identify *components* in the Petri net that can be implemented using BPEL `sequence`, `switch`[3], `pick`, `while` or `flow` components. In case the input graph contains irreducible fragments, these are suggested to be translated manually. Eshuis and Grefen apply a similar strategy in [40] but rule out irreducible graphs as input for their algorithm. Better readability is said to be the benefit of identifying control flow blocks in graphs and representing these by corresponding elements in block-structured languages. However, as discussed above, the size of executable BPEL process definitions severely impedes readability so that adequate visualization strategies have to be provided anyway. So, if really needed, this feature still could be added to BPEL implementations that have been derived from B2Bi choreographies according to the CHORCH approach.

There are more approaches that target the translation of graphs into block-structured languages that leverage similar ideas as [92] or [151]. In [215], van der Aalst and Lassen provide exactly the same algorithm as in [92], but embed the strategy in a larger context. Other approaches provide optimizations by incorporating the local partnerLink strategy for irreducible components within a block folding algorithm or by providing a block identification algorithm that produces the same result independent of the order of applying folding rules [152–154, 217].

Finally, note that the mapping of graphs to a BPEL `flow` element with `links` between the individual branches (as proposed in several approaches) is very limited because such BPEL structures must not contain cycles [137, section 11.6.1].

Translation of Non-B2Bi Choreography Languages The most important non-B2Bi choreography language that is a frequent input for deriving interacting processes is WS-CDL. The advantage of using WS-CDL is that control flow definition is basically block-structured. In that regard, the translation of WS-CDL is less demanding than translating ebBP choreographies. On the other hand, some special constructs such as *silent actions* make the translation particularly challenging. To the best of my knowledge there is no approach that produces fully executable BPEL processes from WS-CDL. For example, Mendling and Hafner [110] create interacting BPEL skeletons from WS-CDL specifications. Weber et al. claim in [230] to provide a mapping of WS-CDL to fully executable BPEL processes. However, this approach relies on looking up implementation artifacts (BPEL snippets and WSDL definitions) from a knowledge base. This is an integral part of the translation strategy and hence fully executable BPEL processes are not derived from a WS-CDL definition alone. This is different from CHORCH's approach for translating ebBP choreographies that achieves executability by means of leveraging standard interfaces for consuming business logic. In addition, the work of Weber et al. is similar to CHORCH in

[3]Although the `switch` element has been removed from BPEL version 2, it can be easily emulated.

defining BPEL processes that integrate cross-organizational message exchanges with local business applications. This resembles the concept of *control processes*. However, CHORCH systematically derives control processes based on the information given in ebBP specifications and hence ensures strict conformance of control processes to the given choreographies. Conversely, Weber et al. use patterns looked up from some knowledge base which does not ensure conformance. In [102], Madiesh and Wirtz propose a development model for deriving abstract BPEL processes from WS-CDL that incorporates compatibility (between interacting BPEL processes) and conformance (of BPEL processes to WS-CDL) checking phases. However, they also rely on manual completion of the generated BPEL processes. Some additional approaches focus on introducing QoS properties into WS-CDL choreographies. The WS-CDL-to-BPEL approach of [102] is proposed to be extended with security annotations by Madiesh and Wirtz in [103]. This is somewhat similar to the specification of security requirements in ebBP BT configurations. The difference is that Madiesh and Wirtz specify security requirements in terms of implementation primitives such as *“usernameToken, signature and encrypt”* [103] whereas ebBP BT configurations contain intentional security goals such as authentication or confidentiality. This difference reflects the slightly differing abstraction layers of ebBP and WS-CDL. ebBP is semantically close to business process models whereas WS-CDL is closer to orchestration models. Furthermore, Madiesh and Wirtz use a proprietary format for specifying choreography security requirements which impedes applicability in practice to some extent. Rosenberg et al. [169] also extend WS-CDL with QoS attributes. However, the set of QoS attributes they support targets performance metrics which is fundamentally different from the QoS attributes contained in ebBP BT configurations. This is also reflected in the style of QoS attribute processing. The performance attributes of [169] typically are *monitored* for particular services which may result in replacing these services (using approaches like [118]). Conversely, the security and reliability attributes of ebBP BT configurations are rather *implemented* for the corresponding services (cf. appendix C). Finally, it is worth noting that none of the above WS-CDL approaches comes up with a formalization of choreography models, formalized execution semantics or the definition of validity criteria that ensure executability of models. Admittedly, as the control flow definition primitives of WS-CDL are more constrained than ebBP primitives the need for formalization is less pressing for WS-CDL.

Beyond WS-CDL, BPEL4Chor and Let’s Dance have been used for deriving interacting BPEL processes. For the discussion of the corresponding approaches [32, 244] please see the previous section.

Approaches Dedicated to B2Bi Scenario Execution This paragraph is dedicated to approaches that use dedicated B2Bi artifacts as input for deriving interacting BSIs.

In [36], Dogac et al. report on one of the first ebXML execution frameworks. The *B2Bi server* of [36] takes ebBP version 1.x and CPPA definitions as input and performs these using ebMS version 1. However, the approach is rather informal compared to CHORCH. There is no definition of process-based choreography implementations using orchestration definitions or anything similar. Hence, a lever for checking the conformance of the BSI implementations to the choreography definition is missing. The formalization of choreography models, a formalized execution semantics or a characterization of model validity is consistently missing as well. Architecture-wise, Dogac et al. also propagate the isolation of the control flow logic from the business logic, but they do not provide standard interfaces that enable a clean separation as CHORCH does.

Khalaf presents an approach for implementing RosettaNet PIPs based on interacting BPEL processes in [74, 75]. As PIP definitions resemble ebBP BT definitions very closely (RosettaNet even provides ebBP representations for several PIPs), this work is highly relevant for the ebBP execution model of section 4.3 and the corresponding implementation in section 5.2. However, Khalaf actually does not provide an abstract execution model with formalized execution semantics. Moreover, she does not validate assumptions about the QoS properties provided by a Web Services-based communication channel as CHORCH does. Instead, she proposes to start out with BPEL templates (underspecified abstract BPEL processes) that capture the synchronous/asynchronous and One-Action/Two-Action distinction of PIPs (cf. RNIF [170]). As RosettaNet decided (after Khalaf's papers) to implement no new Two-Action PIPs and has converted existing Two-Action PIPs into corresponding One-Action PIPs, the One-Action/Two-Action distinction is obsolete today. The integration partners are supposed to create fully specified abstract BPEL processes as refinements of the BPEL templates corresponding to the PIP configuration parameters (almost identical to BT configuration parameters). Finally, executable BPEL processes are supposed to be created. This approach of starting out with interacting BPEL templates is different from CHORCH in not providing an abstract representation of a BT as CHORCH does. Yet, such an abstract representation facilitates communication about the interaction requirements. In addition, CHORCH's execution model provides a lever for deriving fully compliant BPEL processes. As the soundness of the execution model has been validated by means of model checking (cf. appendix D), compatibility between the interacting BPEL processes of CHORCH BT implementations can be taken for granted. Khalaf also identifies the problem of process compatibility and suggests corresponding analysis, but she does not perform the analysis. Furthermore, Khalaf's approach is highly manual whereas CHORCH proposes an automatic translation procedure. Finally, Khalaf also proposes to define an interface for integrating the interacting BPEL processes with business applications. However, she does not come up with a clean interface proposal except for mirroring the WSDL interfaces that the BPEL processes use for cross-organizational communication. Simply mirroring these interfaces leaves out the potential of enhancing BSI facilities with cancellation features as CHORCH does.

In [79], Huemer and Kim propose a model-driven process for creating BPEL orchestrations from ebBP BTs and BCs. Furthermore, they provide some guidelines for the BPEL mappings. However, the mapping for the *BT requester* and *BT* *responder* roles is rather simple as RAs and AAs are not considered. For the mapping of BCs, they suggest the use of BPEL's `flow` construct together with control `links` that reflect the respective BC's graph structure. Hence, this mapping is limited to acyclic processes. Furthermore, they do not provide formalized choreography models, formalized execution semantics or a characterization of model validity. Architecture-wise, the work of Huemer and Kim constitutes an early precursor of the integration architecture proposed by CHORCH. The modularization of BPEL processes according to the nesting hierarchy of ebBP BCs is suggested as well as the separation of control flow logic from business logic. However, the architecture is not fully worked out as has been done in CHORCH later on.

In [61], Hofreiter et al. present an approach for mapping UMM BTs to BPEL. The concept is similar to CHORCH's translation of ebBP BTs into interacting BPEL processes. CHORCH's proposal outperforms the approach in [61] in terms of a thorough analysis of available Web Services QoS features and in offering a formalized, model-checked execution model together with a corresponding formal execution semantics. Furthermore, the CHORCH BT execution model allows the backend systems to cancel a running BT unless the BT is just coordinating upon transaction success. Finally, the model of Hofreiter et al. depends on timing for determining transaction success. The sender of the last business signal determines *success* after having waited until the latest valid point in time for sending the signal has been reached. In between, the opposite party still may send an exception that implies failure of the transaction. However, as there are no strict assumptions about the communication channel, this exception message may be delayed. Hence, the two interacting parties may end up in different states. Admittedly, this is an unlikely, but nonetheless possible, scenario.

Finally, the work presented in [59] that has been discussed as part of the previous section is of relevance for this paragraph.

Alternative Execution Concepts This paragraph contains approaches that provide alternatives to particular aspects of CHORCH's choreography execution concept.

A central XML hub instead of interacting BPEL processes as implementation of a B2Bi choreography is proposed by Trappey et al. in [206]. Such an approach simplifies the derivation of choreography implementations because a central entity controls the progress of the interactions and hence state alignment is not as critical. Yet, Trappey et al. do not define any kind of process-based implementation of B2Bi choreographies. So, a major drawback of the approach in [206] is the missing lever for ensuring compliance of the implementation with the corresponding choreography. Moreover, a central entity that executes a B2Bi choreography is not an option in many real-world settings because no central IT infrastructure is available or even wanted. This is reflected by B2Bi community deliverables such as RNIF [170] and by the B2Bi approaches discussed above. Eshuis et al. also provide a central hub for *animating ebXML transactions* [39]. The focus of this approach is put on simulation rather than production level execution of B2Bi choreographies and hence is not really comparable to CHORCH. Technically speaking, the solution of Eshuis et al. may be extensible to a full-fledged execution framework. Organizationally speaking, the inappropriateness of central IT infrastructure disallows this direction.

Several other approaches target the implementation of classical transaction protocols for ensuring consistency between integration partners [82, 83, 96]. This is different from CHORCH's concept for synchronizing the state of BT requesters and BT responders. The BT execution concept relies on the combination of reliable messaging, synchronous communication and an application level message exchange protocol for ensuring consistency. All message types of the BT execution model already are available in the B2Bi domain which fosters compatibility with existing implementations. Applying a transaction protocol to a BT execution that consists of several application messages without modifying the set of application level messages may be doable (a similar concept has been proposed in preliminary work [185]). Therefore, the SOAP level messages exchanges would have to adjusted and the WS-Coordination [141] and WS-BusinessActivity [140] standards provide functionality for that. However, the large variety of business application products used in B2Bi scenarios bears the risk of producing an interoperability nightmare if this approach was followed. Instead, the strategy of completing and correcting existing BT exchange procedures and enhancing these with reliability features that are applied per application message transmission and do not span several application messages was followed for CHORCH.

Finally, Singh introduces *Local State Transfer (LoST)* in [202] as execution framework for BSPL choreographies (introduced in section 2.3). LoST is significantly different from all the above execution approaches that leverage a rather imperative style of control flow definition. Conversely, LoST *proxies* accept messages of communication partners or of business applications depending on the locally available parameter values that are defined as BSPL message components. For example, if the value of an `in`-parameter of a message to be transmitted on behalf of the business application was missing in the LoST adapter, then the LoST adapter would throw an error because the `in` adornment requires that the parameter must be available beforehand. Essentially, LoST adapters enforce BSPL protocols rather than executing them. Business applications and communication partners drive the protocol by submitting messages. In such a scenario, it would be interesting to see how timeout management similar to event-based choices is performed. For the communication channel in use, Singh makes the assumption that any message will eventually be delivered which could be implemented using a lower-level protocol. This could result in duplicate application level messages but as parameter values are bound only once, duplicate message transmissions do not do any harm. All in all, LoST is interesting because it respects the characteristics of distributed systems very closely, that means a protocol can only progress if the relevant parameters have been communicated beforehand. Hence, progress cannot be made without having sent a message. However, it also requires that this distributed system property becomes somewhat evident in the protocol definition. This, in turn, requires the protocol designer to configure the message parameter dependencies correctly. Whether or not this is easier than specifying the control flow of B2Bi interactions directly in terms of imperative processes is still to be investigated. For CHORCH, the strategy was taken to provide choreography modelers with concepts they already know and then to implement it on available IT infrastructure even if this may result in more complicated implementation constructs than those leveraged for LoST.

7.4. Multi-Layer Approaches

This section discusses approaches that are relevant throughout several layers of the CHORCH approach.

In [20], Bianchini et al. present the *from process to services* (P2S) approach for turning their BPMN collaborations-based model of an interorganizational system into Web Services in several steps. Therefore, a data dependency and control flow graph derived from a BPMN model have to be semantically annotated and then heuristics are used to identify candidate composite services for implementing the BPMN activities. Metrics like coupling and cohesion are used to evaluate the derived service design. The work of Bianchini et al. (and similar approaches) is different from CHORCH in leveraging semantic technologies for service identification and design. The derivation of implementation artifacts such that compliance to choreography models or compatibility between interacting processes is ensured is of minor concern. The implementation of coordination protocols is consistently not elaborated.

In [10], Barros et al. propose an approach for *Multi-staged and Multi-viewpoint Service Choreography Modelling*. Role-based, milestone-based, and scenario-based views on choreographies are provided in [10]. The *role-based view* is used to capture the relationships between roles leaving out the control flow perspective. This corresponds to the information typically contained in BPMN conversation diagrams. Then the authors suggest to define so-called *milestones* to represent the major states of the choreography while leaving out the details of necessary message exchanges that are needed for reaching these milestones. In an additional step, these details can be completed. In so far, the work of Barros et al. reflects the concept of ebBP-ST choreographies where *shared states* represent the progress of choreographies and BTs and BCs are executed to reach these. The approach in [10] also significantly resembles a precursor of ebBP-ST that I have published in [186] one year before [10]. In [186], so-called *micro-choreographies* are used to abstract from the message exchanges that are necessary for reaching shared states (milestones). However, BPMN conversation style diagrams for capturing role relationships from a static point of view are not part of [186].

In [204], Telang and Singh use so-called *commitment-based modeling* to capture the business model of PIP patterns (and hence ebBP/UMM transaction patterns). A commitment characterizes an obligation that some role commits to some other role if some precondition becomes true. For example, the exchange of a quote request can be used to establish the commitment of a seller to provide some product if the customer provides the payment. Furthermore, the creation of additional commitments can be the obligation of a commitment as well. Modeling commitments and relationships between commitments allows for modeling the value exchange relationships between multiple roles. For example, a seller may offer the obligation to request shipping from a shipping provider if the customer sells the goods. It is vital to note that Telang and Singh do not define an operational model for exchanging the messages that establish commitments. Instead, they just capture the value exchange relationships between roles so that their approach is relevant for business modeling and not for business process modeling. Consistent with that, Telang and Singh present in [205] an approach for checking the implementation of a business model as defined in [204]. They use UML sequence diagrams for specifying operationalizations of business models and NuSMV for checking compliance of the operationalizations to the business models. This clearly shows that commitment-based modeling of BTs is a complementary approach to CHORCH and it would be interesting to investigate the integration of commitment-based modeling with CHORCH.

In [86], Kramler et al. also propose to capture the business model of interactions first. Although they focus on Web Services collaborations in general, their approach is highly relevant for B2Bi as well. Kramler et al. propose to first capture the business model of interactions using UML 2 collaboration diagrams which is somewhat similar to the work of Telang and Singh which has been developed some years later. Then, Kramler et al. define an operational refinement of the business model using UML activity diagrams. Finally, the implementation of these activity diagrams is to be specified using UML scenario diagrams. The last two types of models of [86] are similar to deriving BPEL implementations of B2Bi choreographies as CHORCH does. However, Kramler et al. do not define mappings for this transition, nor do they provide a formalization of models, a characterization of validity or a formalized execution semantics. Furthermore, there is no obvious way how the approach in [86] could be used to derive fully executable BPEL processes. On the other hand, Kramler et al. outline the modeling of the data perspective of interactions and a concept for capturing the business model of choreographies which is not in the scope of CHORCH.

In [63], Huemer et al. propose the *Business Semantics on top of process technology* (BSopt) approach that is similar to [86] in advocating the integration of B2Bi models throughout several abstraction layers. In contrast to [86], Huemer et al. rely on UMM for capturing B2Bi choreographies which bears the promise of wider adoption by the B2Bi community because UMM provides B2Bi domain concepts and is promoted by UN/CEFACT. Huemer et al. also provide a concept for business modeling as well as a data perspective on B2Bi choreographies which is not part of CHORCH. The content in [63] is rather an overview of the BSopt approach so that a comparison with this thesis is difficult. However, all known contributions of the authors have been discussed above. The distinctive features of CHORCH are a thorough investigation of the message channel qualities that can be assumed for Web Services communication, a validation of the BT implementation based on model checking, a mapping for BC collaborations and not only for BTs as well as the formalization of models and execution semantics together with the characterization of model validity.

Finally, the PhD thesis of Marco Zapletal [247][4] has significant overlap with this study. Firstly, Zapletal defines a state machine based execution model for UMM BTs. As UMM and ebBP BTs are almost identical, his execution model is directly relevant to the execution model defined in section 4.3. The most important differences to CHORCH are the following. CHORCH provides a formalized execution semantics that is aligned with the communication qualities of Web Services technology. Furthermore, the execution model of CHORCH has been validated by means of model checking which guarantees that the interaction partners always produce consistent outcomes, that means either both partners (eventually) result in a success state or in a failure state. In addition, CHORCH's execution model allows for canceling running BTs whereas the execution model in [247] does not. Occasionally, the execution model in [247] may produce inconsistent outcomes. Consider the exemplary *notifier* state machine in [247]. The notifier determines success some time after having sent the last business signal (called processing acknowledgment which corresponds the acceptance acknowledgment of ebBP BTs). Therefore, the timeout value for sending the processing acknowledgment is added to the time of receipt of the last business document. When this deadline has been reached, the notifier state machine determines success. *"Until this time the notifiee - as the responder within the transaction - may still signal a time-out exception or a failed business control"* [247]. However, such a failed business control message may be delayed and the notifier may determine success before the message has been received. According to the state machine definitions, the notifiee would then end up in a failure state nonetheless so that both partners would fail on agreeing on the BT outcome. It is vital to note that this does by no means render the contribution of Zapletal useless because the probability of having such an error sequence is not very high for well-behaving systems. In the end, the user must decide whether having inconsistent outcomes in rare cases is acceptable or not. Architecture-wise, the state machines of Zapletal correspond to CHORCH's *control processes* and hence also facilitate separation of control flow logic from business logic. In contrast to [247], however, CHORCH also takes this concept to the level of BCs and defines mapping rules for creating higher-level control processes. In addition, CHORCH provides formal definitions of B2Bi choreographies, their formal execution semantics and a characterization of valid models. On the other hand, the work of Zapletal also comprises an approach for business modeling and the data perspective of B2Bi choreographies. So, in practice, users may want to combine the strengths of both approaches which would essentially result in an integration of CHORCH with UMM.

To summarize, the discussion of this chapter shows that the CHORCH approach is different from existing work in combining the strengths of dedicated B2Bi approaches/languages with formal rigor. While existing formal approaches to choreography specification and execution lack B2Bi domain concepts, existing B2Bi approaches and languages lack formal underpinning. CHORCH provides the B2Bi choreography

[4]The contents of [65, 246] are reflected in his thesis and are therefore not discussed separately.

modeler with the domain concepts she is used to and defines unambiguous formal semantics as well as validity notions on top. Based on CHORCH's B2Bi choreography model, execution and analysis support is provided that goes beyond existing approaches. In addition, a tool-chain is defined that allows for precise semantics of even abstract visual B2Bi choreographies that subsequently can be enriched with technical parameters and finally be turned into services-based orchestrations without violating compliance with the initial control flow agreement. Throughout all abstraction levels under consideration, only international standard languages and formats are used for specification and implementation. All these facts taken together make CHORCH unique with respect to the current scientific landscape.

The next chapter gives a more detailed summary of this thesis.

8. Conclusion and Future Work

The overall research question of this thesis postulated in section 1.2 asks for *"how B2Bi choreographies can be used as implementation contracts for services orchestration based B2Bi systems?"* This question then has been split up into the three more manageable questions that have been elaborated on mainly in chapters 3, 4, 5 and 6. In what follows the contributions made by this thesis during working out the respective research questions are presented (see also section 1.3):

- **RQ 1.1 What is a suitable B2Bi choreography language?** The rough scope for answering this question is given by the type of B2Bi systems that this thesis is dedicated to as outlined in section 1.1. In chapter 3, the core requirements for B2Bi choreography languages are discussed and some initial design choices such as the usage of ebBP as choreography specification language and BPEL as orchestration language are discussed. Building upon these requirements, chapter 4 presents this thesis' proposal for a suitable B2Bi choreography language that is based on the ebBP standard. The ebBP standard constitutes an implementation technology agnostic choreography format that offers B2Bi domain concepts. This thesis then amends the ebBP concepts where necessary and defines ebBP-ST, ebBP-Reg and SeqMP as distinct B2Bi choreography styles that address conflicting requirements of chapter 3. The most important amendment of ebBP is the definition of the BT execution model together with formalized execution semantics described in section 4.3. Formalized execution semantics also are given for the identified B2Bi choreography styles. Furthermore, validity criteria for the choreography styles are defined that ensure either translatability into BPEL based implementations according to the definitions of chapter 5 and appendix B (ebBP-Reg, ebBP-ST) or amenability to analysis using the framework defined in section 4.6 (SeqMP).

  The core contribution of this thesis concerning choreography definition is twofold: Firstly, a formal operational execution semantics is defined for ebBP models that covers BTs as much as BCs. Such a semantics has not been available so far. Secondly, the identified B2Bi choreography styles allow for almost unconstrained definition of control flow between the BTs and BCs of a choreography. The basis for this is using state machines as underlying control flow definition paradigm. The applicability of this rather simple paradigm is backed up by the requirements definition of chapter 3 that reveals the coverage of the overwhelming majority of targeted B2Bi scenarios by the identified choreography styles. This is different from other choreography approaches that either constrain the available control flow primitives significantly, for example, require block-structuredness, or focus on control flow primitives that are of minor importance to the B2Bi domain.

- **RQ 1.2 To what extent and how can services orchestrations be used to implement valid B2Bi choreographies?** Again the rough scope for this question as well as the relevant requirements are given in section 1.1 and chapter 3. Furthermore, the B2Bi choreography styles given in chapter 4 define the *"valid B2Bi choreographies"* to be implemented using services orchestrations. Based on the integration architecture described in section 4.2, chapter 5 describes how Web Services and BPEL technology can be used for the implementation of B2Bi choreographies. In particular, the mapping rules for deriving BPEL processes from ebBP-Reg as the more demanding binary B2Bi choreography style are given (a mapping algorithm for ebBP-ST developed during a diploma thesis project is given in appendix B). There is no mapping for SeqMP choreographies because this multi-party choreography style is not supposed to serve as implementation contract, but rather for the purpose of analyzing synchronization deficits in advanced multi-party scenarios.

  The focus of the choreography implementation described in chapter 5 is put on supporting the identified B2Bi choreography styles while leveraging dedicated interoperability technologies like Web Services and BPEL as well as an integration architecture based on *control processes*. Therefore, the contribution of this thesis in that regard is not outperforming established B2Bi systems in terms of traditional communication qualities such as throughput, but the following:

  Firstly, the integration architecture based on control processes separates the control flow logic from the application logic of B2Bi choreography implementations. This allows for the derivation of fully executable BPEL processes that govern the cross-organizational message exchanges while importing business logic from business applications when needed. Furthermore, the integration architecture is organized in a modular manner that reflects the hierarchical organization of ebBP choreographies. This conceptually enables leveraging established B2Bi communication protocols such as AS2 for selected business document exchanges by wrapping the corresponding messaging subsystems using Web Services technology.

  Secondly, the mapping from ebBP to BPEL is unique in allowing for irreducible loops within the choreography input graphs and in producing fully executable BPEL process definitions (cf. chapter 7).

  Thirdly, the implementation strategy for B2Bi choreographies combines synchronous and asynchronous coordination (cf. section 2.1) in an innovative way where messages are transmitted synchronously and the processing of messages is performed asynchronously. In this regard, it is worth noting that the availability of advanced security goals such as mutual authentication as well as reliability is analyzed for Web Services technology. The availability of these communication qualities on the transport channel obviates the need for incorporating intricate security and reliability mechanisms at the application level. Taken together, synchronous communication and the availability of relevant security and reliability on the transport channel greatly simplify the design of the given interaction procedures that guarantee consistent results of B2Bi choreography executions across integration partners. This is different from other approaches that assume either asynchronous communication or do not check the available communication qualities of the selected communication technology at all. Frequently, the problems of realizing B2Bi security and reliability goals are just ignored in those approaches. Finally, it is vital to note that technical control messages are used to serialize concurrent requests for BT executions. This enables the use of control flow structures that are forbidden in other approaches (cf. chapter 7).

- **RQ 1.3 How can B2Bi choreographies be visualized?** The answer to this question builds upon the results of chapter 4 and defines an abstract visual notation for CHORCH's B2Bi choreography styles based on the BPMN choreography notation. In order to adapt BPMN choreographies to B2Bi choreography modeling, the semantics and labeling rules for several BPMN constructs are changed. In addition, a significant set of BPMN constructs not needed for representing B2Bi choreographies is forbidden. It is vital to note that no new modeling symbols are introduced which allows for leveraging standard tools for modeling CHORCH's B2Bi choreography styles in BPMN.

  The contribution of this thesis in that regard is defining a kind of B2Bi profile for BPMN choreographies that offers B2Bi domain concepts such as BTs and disregards unnecessary modeling primitives. Section 6.3 further shows that ebBP models can be derived from such visual models without affecting the control flow definition. Thus, this thesis' B2Bi choreography visualization allows business users to focus on the type and sequence of business document exchanges while the corresponding ebBP models then allow software engineers to complete the relevant technical configuration data.

Splitting the research problem of this thesis up into three research questions is methodologically underpinned by the relationship of the research questions to the abstraction layers as depicted in figure 1.3 of section 1.1.2. The first research question targets the choreography abstraction layer and the relevant concepts and semantics for defining B2Bi choreographies. The second research question asks for how the identified B2Bi choreographies can be integrated with the lower level orchestration layer and the third research question asks for how the identified B2Bi choreographies can be integrated with more abstract perspectives on business document exchanges.

Beyond the formulation of the research problem as explicit research questions, section 1.3 characterizes it by defining a *goal state* of B2Bi technology. In what follows, the reachability of this goal state using the results of this thesis is discussed:

- *The barrier for adopting process-based B2Bi is lowered.*
  The first factor for lowering the barrier is enabling the usage of low-cost Internet protocols. This thesis provides guidelines on how to use Web Services technology, which leverages Internet protocols, for implementing B2Bi choreographies. The analysis not only concerns the definition of BPEL based processes that orchestrate individual Web Services calls, but also treats the implementation of advanced communication qualities that are needed for B2Bi settings such as mutual authentication or reliable messaging. In particular, the implementability of the *Secure WS-ReliableMessaging Scenario* [4,44] using Web Services technology is checked. While such advanced features are available for the selected platform (cf. section 5.1 and appendix C), they cannot be assumed to be available in heterogeneous settings, i.e., when integration partners make use of WS stacks of different vendors (see discussion of section 5.1, appendix C and [183,196,197]). In this regard, the concern of Werner Vogels (CTO of Amazon.com and core designer of the Amazon Web Services cloud facilities) highlighted in 2003 seems to become reality: *"The greatest threat to Web Services' large-scale success is vendor politics"* [218]. Hence, in practice, B2Bi integration partners who want to leverage Web Services technology will have to stick to Hypertext Transfer Protocol Secure (HTTPS) for securing the communication channel and running a reliability layer (for example, based on WS-ReliableMessaging) on top of that. Beyond security and reliability qualities, B2Bi based on Web Services may be problematic if very large business documents or binary business content is supposed to be exchanged. In this case, the use of DOM-based XML libraries may result in memory errors or the attachment mechanism for transporting binary data may not be interoperable across different WS stacks. Then, at least Web Services can be used for the coordination of the control flow between individual BTs and established B2Bi communication protocols such as AS2 can be used to perform the business document exchanges.

  The second factor in lowering the barrier for adopting process-based B2Bi is the availability of a comprehensive approach supporting B2Bi choreographies from visual models through technical configuration to process-based implementations. This is exactly what this thesis provides. In particular, existing standards are amended and clarified where needed, unambiguous execution semantics are defined and model validity is characterized.

  So, for this thesis the claim is made that the barrier for adopting process-based B2Bi is indeed lowered even if immediate applicability to concrete B2Bi projects requires the development of production quality tooling.

- *B2Bi choreographies can be used as standardized, technology-agnostic contractual obligations between integration partners.*
  From the perspective of the technologies used, this thesis relies exclusively on standard languages, namely BPMN, ebBP and BPEL. The BPMN and ebBP models are indeed technology agnostic. Furthermore, the definition of

ebBP-Reg and ebBP-ST shows how to use choreographies as implementable contractual obligation. Moreover, chapter 5 describes how an implementation can be derived in an automated manner by means of ebBP to BPEL mapping rules. Admittedly, these rules imply several assumptions about the messaging configuration options such as Web Services bindings and endpoints. However, the goal of this thesis in that regard is not the development of a comprehensive execution framework that takes all kinds of different messaging options into account, but showing that the identified binary B2Bi choreography styles can be implemented using services orchestrations.

In a practical setting, a format for managing messaging configuration data would be needed. The CPPA standard in version 2 [128] provides partial support for this, but it is tied to ebMS too tightly for really being of use for Web Services based integrations. Version 3 [127] of CPPA promises to offer the required functionality, but it is still in an immature state. During this dissertation project, some initial results have been produced for how CPPA can be used for specifying the relevant data [188], in particular for incorporating alternative B2Bi communication technologies. However, a comprehensive treatment of all relevant communication protocols as well as prototypic implementations would be required for practical settings. It is vital to note here that such a comprehensive aim calls for a concerted project of major B2Bi communities and B2Bi adopters.

- *B2Bi is facilitated by an integration architecture that enables separation of the business logic from the control flow logic.*
  The separation of the business logic from the control flow logic of a B2Bi scenario is facilitated by the integration architecture described in sections 4.2 and 5.1. *Control processes* represent the central concept for enabling this separation. They are responsible for performing the cross-organizational message exchanges according to the agreed-upon choreography definition and import existing business logic from business applications using standard interfaces. These interfaces, in turn, are derived based on a fixed set of technical control messages as well as the BT configuration data and BC definitions of a given choreography. As control processes do not contain business logic, they can be generated as completely executable BPEL processes. Automated generation facilities (based on the assumption that the corresponding mapping rules are correct) ensure compliance of control processes to choreography definitions because the generated process definitions do not have to be modified manually afterwards (as opposed to other approaches, cf. chapter 7).

While any kind of business logic can be separated from *control processes*, the connected business applications necessarily contain elements of control flow logic. This concerns, for example, detecting the need for triggering the execution of a particular BT or BC. Such an activity bears aspects of both business

logic and control flow logic. On the one hand, the assessment whether or not a new business document exchange is to be triggered clearly is a business decision. On the other hand, when interpreted as an event, triggering a business document exchange is a concern of control flow logic. Moreover, dependencies between business document exchanges such as the availability of order line items for creating invoice line items may be controlled by business applications which affects control flow logic as well.
In this regard, it is vital to note that even if business applications may contain aspects of control flow logic and may even take over the task of displaying admissible control flow paths to the application user, they still cannot change the control flow of the cross-organizational message exchanges. This is a logical implication of interposing control processes in the communication between business applications of the integration partners. Whatever kind of control flow logic is encoded in the business applications, it cannot modify the agreed-upon cross-organizational message exchange flow as implemented by the control processes. This concept moves the process interoperability problem from the *unsafe environment* of cross-organizational communication to the *safe environment* of communication between control processes and business applications (which is performed using the IT resources of one partner).

- *B2Bi is facilitated by an integration architecture that enables tracking the progress of business interactions.*
  Process-based implementation of BSIs is an important enabler for tracking the progress of business interactions. Firstly, the progress of the interactions can be interpreted relative to the overall process model. Secondly, the log messages that are needed for collecting progress data can be created such that log messages make sense from a process model perspective. Otherwise, it may be hard to associate log message contents with the process model constructs. State machine based process models as defined in this thesis are particularly well suited for tracking progress because the corresponding states can be used as point of reference. In that regard, it is worth noting that the execution of a BT or BC is to be interpreted as a state of choreography models as well. Furthermore, the hierarchical organization of ebBP-Reg allows for aggregated states (that means BCs) that represent several lower level states (that means the BTAs and BCAs of the respective BC).

  In [54], a method that relies on the results of this thesis for generating BT control processes such that progress messages are reported to a central monitoring service is presented. This demonstrates the feasibility of tracking progress using automated procedures. However, it would still require considerable effort to implement an analogous approach for complete ebBP-ST and ebBP-Reg models. Moreover, aggregated performance data such as maximum, minimum or average execution times of the individual activities of a choreography definition would have to be taken into account.

- *Model-driven technologies can be used to derive orchestrations that govern the control flow of message exchanges.*
  This goal predominately is supported by this thesis based on the mapping from ebBP to BPEL as described in chapter 5. This is indeed a model-driven approach. The computation independent model of a BT or BC is specified more precisely using the execution models for BTs and coordination protocols for BCs (platform independent model) that then can be translated into a Web Services and BPEL based implementation (platform specific model). The validity of the translation procedures given in this thesis has been demonstrated by manually performing the proposed translation rules and by implementing part of the translation rules for ruling out manual bias [54, 182].

  However, B2Bi partners still may be reluctant to adopt such a model-driven technology for reasons of maintaining existing implementation approaches, enabling low-level process customizations or performance optimizations. In this case, the mapping rules are still beneficial for generating implementation artifacts that then are used as reference for the actual, hand-coded implementation or for testing purposes, for example, as part of a testbed.

  Moreover, leveraging the proposed mapping rules in a practical model-driven approach would require the incorporation of CPPA like technical configuration data as discussed above.

  Another characteristic of model-driven technologies is that more than one platform specific model may be leveraged. This issue has not been elaborated on in this work because the core focus was put on the choreography definition. The translation to implementation artifacts is more of a validation of the choreography models than the actual research problem. Yet, the results of a master thesis supervised during this dissertation project [93] show that Windows Workflow[1] may be eligible as alternative implementation platform because it is similarly expressive as BPEL. However, the generation of Windows Workflow based BC implementations is still an open research problem. Moreover, potential interoperability problems between BPEL-based and Windows Workflow-based B2Bi choreography implementations calls for additional research.

Compared to the derivation of BPEL implementations, the transformation from BPMN models to ebBP models as suggested in chapter 6 is more of a format conversion. The manual derivation of ebBP processes from the BPMN use cases developed for the *RosettaNet Methodology for Creating Choreographies* [173] reveals that the semantic gap between both choreography representations is characterized by technical configuration data. However, the control flow definition remains unchanged. It is worth noting at this point that the two representations thus enable business users to focus on the type and sequence

[1]`http://msdn.microsoft.com/en-us/netframework/aa663328`, last access: 12/20/2011

of business document exchanges while software engineers then are allowed to complete the relevant technical configuration data.

- *B2Bi models can be stringently visualized without a significant amount of technical detail.*
  This goal is catered for by this thesis in chapter 6. Based on the B2Bi choreography styles defined in chapter 4, the BPMN choreography notation is used to provide an abstract view on the corresponding ebBP models. All technical configuration data such as the exact business document versions or security and reliability attributes are left out. Moreover, there are no implementation technology dependent artifacts like WSDL interfaces, endpoint addresses or protocol bindings.

  The BPMN based B2Bi choreography visualization is stringent for two main reasons. Firstly, there are informal, yet precise rules for creating complete and valid choreography models (cf. section 6.2) that correspond to the ebBP based B2Bi choreography styles proposed in this work. Secondly, the derivation of ebBP models does not change control flow (cf. section 6.3) and hence the generated classes of ebBP models have the unambiguous semantics of ebBP-ST, ebBP-Reg and SeqMP.

  BPMN has been used as visual notation because the B2Bi choreography visualization was developed in the context of the RosettaNet MCC project and representing complete choreography definitions in one diagram was a major requirement. Instead of BPMN, UMM could have been used as visual notation. Indeed, UMM is a dedicated B2Bi notation and therefore outperforms BPMN in terms of offering domain concepts. However, it requires multiple views to be defined (cf. section 2.3.3). While this was a barrier in the MCC project, it should be a feature for large-scale B2Bi projects. In particular, the information modeling view of UMM for focusing on the content of the exchanged business data is relevant. The explicit view on business information allows for checking the control flow definitions against the business data that is exchanged which fosters detection of modeling errors. In so far, the application of UMM visualization features to CHORCH's choreography styles is a logical extension of this thesis. Note that this would also be in line with [68] that identifies *Multi-perspective modeling* and *view integration* as important business process modeling concerns.

  Finally, it is worth noting that optimizing usability of the visualization was greatly limited by selecting CHORCH's B2Bi choreography styles as reference on the one hand and BPMN choreographies as notation on the other. This significantly constrains the freedom of selecting usability-optimized shapes and composition rules so that an empirical usability study has not been performed.

Beyond the desirable extensions of this thesis highlighted throughout the above discussion, the following research problems have not been considered.

The performance impact of using the proposed control process based integration architecture as well as Web Services technology is not thoroughly analyzed. It seems to be obvious, though, that there is a performance impact (cf. discussion in the introduction of chapter 5). To not compare the performance of the proposed implementation guidelines for B2Bi choreographies to traditional B2Bi messaging solutions is acceptable for two main reasons. Firstly, the primary purpose of deriving a BPEL based implementation for B2Bi choreographies is validating the implementability of the B2Bi choreography styles of this thesis and not providing a generic execution framework that serves all kinds of B2Bi scenarios. Secondly, the proposed integration architecture has a clear focus on improved functionality compared to more or less process-agnostic BSI implementations and not on performance improvements. So, an implicit assumption of this thesis is that there is a class of B2Bi scenarios for which the expected performance impact of using the CHORCH approach is outweighed by enhanced visibility and improved control.

As there apparently is a trade-off between performance and functionality, an interesting open research issue is the development of a decision model for selecting between the CHORCH approach that guarantees process compatibility and enables high process visibility on the one hand and more traditional B2Bi solutions that foster high throughput but tend to suffer from vendor lock-in on the other.

Finally, the formalization of the execution semantics for CHORCH's B2Bi choreography styles provides the basis for creating simulation features. These could be used, among others, for explaining the permissible choreography execution traces to unexperienced users and for analyzing the message sequences that may have to be consumed or produced by corresponding implementation artifacts.

The above discussion shows that the CHORCH approach indeed can be said to improve the state of the art of B2Bi by providing a top-down style framework throughout several abstraction layers. BPMN choreography notation can be used to abstractly capture B2Bi choreographies. Such visual representations then can be turned into corresponding ebBP models that allow for adding technical parameters such as business document versions or security and reliability attributes. Finally, these ebBP models can be transformed into BPEL-based implementations.

The validity of this thesis' results is not only given by following rigorous research methods and external validation through workshop, conference and journal program committees. It is also reflected in the acceptance of core results in two international RosettaNet standards:

- The execution model for BTs (section 4.3) as well as corresponding implementation guidelines of chapter 5 have been contributed to the *RosettaNet Message Control and Choreography (MCC) - Profile-Web Services (WS), Release 11.00.00A* standard [172].

- The visualization of B2Bi choreographies using BPMN choreography notation of chapter 6 has been contributed to the *RosettaNet Methodology for Creating Choreographies, R11.00.00A* standard [173].

Beyond validity of research results, the cooperation with RosettaNet also fosters timely dissemination of research results. The quality of the research results is also reflected by a linkedin recommendation[2] of Dale Moberg, the lead editor of the ebBP specification and co-worker within the RosettaNet MCC team, on my contributions to the *RosettaNet Methodology for Creating Choreographies* [173]:

> *"[..]The result [of Andreas Schoenberger's work] was an exceptional blend of theory and practice, and will serve as the basis for future business process management initiatives for RosettaNet. It will be consulted by other initiatives seeking to enhance process descriptions with QoS, visibility, and manageability information."*

[2]`http://de.linkedin.com/pub/andreas-schoenberger/10/a47/77a`, last access: 12/20/2011

A. B2Bi Requirements Sources and Classification

This appendix gives the associations of B2Bi requirements with source publications, B2Bi challenges and B2Bi schema abstraction layers as referenced in chapter 3.

Tables A.1 and A.2 (pages 290/294) associate each requirement with sources it has been identified in which enables traceability of the work. Moreover, depending on the requirements source, this information may be useful for looking up additional context information, different variants of or even solutions to requirements. In case a requirement has been identified in a requirements source according to the rules defined in section 3.1 the respective cell has been valuated with a “+”. When a requirements source only describes some useful features for satisfying a requirement, but does not explicitly associate these features with a requirement or declare the requirement itself, then the respective cell has been valuated with a “0”. Otherwise, the cell has been left empty.

Table A.3 (page 297) defines the contribution each requirement adds to addressing the B2Bi challenges which helps in assessing the importance of each requirement in different B2Bi scenarios that vary with respect to the importance of each challenge. Moreover, the challenge association is helpful in identifying interrelations between requirements. Clearly, the contribution of fulfilling a particular requirement in addressing a challenge differs from challenge to challenge. For differentiating the level of contribution each of the researchers (cf. section 3.1) determined the function $ctrb : \text{REQ} \times \text{CHL} \rightarrow \{0, 1, 2\}$ (table A.3), where REQ is the set of requirements presented below, CHL is the set of challenges presented in section 3.2 and {0,1,2} represent the level of contribution varying from *no contribution* (0) and *some contribution* (1) to *highest contribution* (2). In order to find the challenge a requirement contributes to most, the constraints

$$\begin{array}{ll} \text{C1:} & \forall (a, b) \in \text{REQ} \times \text{CHL}, ctrb(a, b) = 2 : \\ & \nexists (x, y) \in \text{REQ} \times \text{CHL}, x = a \wedge y \neq b \wedge ctrb(x, y) = 2 \\ \text{C2:} & \forall a \in \text{REQ} : \exists (x, y) \in \text{REQ} \times \text{CHL}, x = a \wedge ctrb(x, y) = 2 \end{array}$$

had to be respected when creating the relation saying that for each requirement exactly one challenge shall be assigned the contribution value 2. In so far, the contribution values for a given requirement depend on each other and therefore the contribution values for two different requirements cannot be compared (therefore the term “highest contribution” has been chosen). The researchers’ relations then have been discussed and merged in several meetings which lead to the result presented in table A.3 on page 297. Constraint C1 is satisfied for each requirement except for requirement 13 (Control flow definition) where both the *feasibility* and the *agreement* challenge have been valuated with 2. Constraint C2 is satisfied for each requirement except for requirement 78 (formal methods) for which no challenge could be found that benefits substantially more from requirement 78 than the other challenges.

Finally, table A.4 (page 300) associates each requirement with the abstraction layers of the B2Bi schema (figure 1.3). This information is helpful for identifying the problem domains affected by a requirement and available solution technologies. Moreover, as these abstraction layers correlate with software development phases this classification helps in deciding when to consider a requirement. The degree to which a requirement should be considered on abstraction layers has been specified by the function $csdr: \quad \mathrm{REQ} \times \mathrm{CHL} \rightarrow \{\%, -, 0, 1, 2\}$ (table A.4), where REQ is the set of requirements presented below, CHL is the set of challenges presented in section 3.2 and {%,-,0,1,2} represent the degree of consideration varying from *not applicable*(%), *should not be considered*(-), *could be considered*(0) and *should be considered*(1) to *is strongly recommended to be considered*(2). Again, each researcher first determined *csdr* individually before the final function presented in table A.4 has been merged during several meetings.

Index	Requirement	[134]	[128]	[136]	[131, 132]	[208]	[123]	[170]	[101]	[3]	[126]	[125]	[100]	[148]	[234]	[187]	[213]	[193]	[176]	[89]	[88]	[243]	[34]	[162]
1	Multi-level and multi-view description					+		+								+	+		+	+				+
2	Support for business transactions	+	+	0		+	0	+																
3	Support for business signals	+	+	+		+		+																
4	Hierarchical decomposition; Composability	+	+			+																+		
5	Support for binary collaborations	+	+			+	+	0																
6	Support for multi-party collaborations	+	+			+																		
7	Support for business documents	+	+	+		+	+	+				+							+					
8	Quality of service	+	+	+	+	+	+	+		+				+						+		+	+	
9	Control flow definition	+	+			+	+	0				+			+							+		
10	Exception/Error handling	+	+	+		+	+	+												+		+		
11	Role modeling	+	+	+		+	+	+														+		+
12	Role mapping	+	+	+		+																		
13	Support for business document attachments	+	+	+		+		+																
14	Support for process version control	+	0		+	+													+				+	
15	Technology independence of process model	+				+	+	+											+			+		
16	Integration partner binding		+	+																				
17	Flexible configuration of transfer/transport protocol		+	+				+				+												
18	Flexible configuration of document exchange characteristics		+	+		+		+																
19	Negotiation of business capabilities		+					+		+										+				
20	Negotiation of communication capabilities		+					+		+												+		
21	Configuration data for runtime systems		+	+				+																

Table A.1.: Sources of Requirements (part 1)

Index	Requirement	[134]	[128]	[136]	[131, 132]	[208]	[123]	[170]	[101]	[3]	[126]	[125]	[100]	[148]	[234]	[187]	[213]	[193]	[176]	[89]	[88]	[243]	[34]	[162]
22	Interfacing with backend systems			0															+		+			
23	Message correlation			+				+				+										+		
24	Communication interface			-																				
25	Registry functionality				+	0				+													+	
26	Repository functionality				+					+									+				+	
27	Metadata definition				+	+		+		+													+	
28	Classification of processes				+	+		+																
29	Definition of associations between processes				+		+																	
30	Cataloging of processes				+			0																
31	Validation				+			+									+							
32	Lifecycle management of B2Bi artifacts; Methodology				+	-											+	+	+	+			+	+
33	Management of relationships among service/process providers and service/process users				0																		+	
34	Process governance				+				+														+	
35	Extensibility				+	+												+		+				
36	Event propagation				+																	+	+	
37	Visual representation	0				+	+	+																+
38	Machine-processable format	+	+	+	+	0			+		+													+
39	State-based modeling					+					+											+		
40	Data formats and data codes						+	+												+	+			
41	External communication						+																	
42	Consistency	+	+	+	+	+	+	+	+							+								
43	Ease of maintenance				+		+																	
44	Ease of explanation						+																	+
45	Auto-generation of artifacts	0	+			0	+									+			+				+	+
46	Description of usage scenarios					+	+	+																
47	Description of business requirements						+	+																
48	Description of business benefits						+																	

Table A.1.: Sources of Requirements (part 1)

Index	Requirement	[134]	[128]	[136]	[131, 132]	[208]	[123]	[170]	[101]	[3]	[126]	[125]	[100]	[148]	[234]	[187]	[213]	[193]	[176]	[89]	[88]	[243]	[34]	[162]
49	Pre/Post-conditions of process/task executions	+				+	+	+	+															
50	Formal methods								+		+					+	+							+
51	Analysis features								+		+					+	+							+
52	Intelligible feedback of analysis								+															
53	Language for semantic constraint specification	0				0	0		+														+	
54	Semantic constraint management								+							+								
55	Semantic constraint violation traceability								+															
56	Reputation information management									+														
57	Language domain appropriateness										+		+						+					
58	Language participant knowledge appropriateness										+								+					+
59	Language comprehensibility appropriateness										+		+						+					+
60	Language technical actor appropriateness										+													+
61	Language organizational appropriateness										+													
62	Reasonable tool support										+								+		+			
63	Traceability between process model and process execution										+					+	+		+					
64	Flexibility by underspecification												+				+	+		+				
65	Adaptability												+					+						
66	Dynamism												+					+						
67	Control flow patterns														+									
68	Data oriented process definition	+				+	0	+							+									
69	Simulation																+							+
70	Usage of standards	+	+	+	+	+	+	+								+	+		+					
71	Process flexibility by design																	+						

Table A.1.: Sources of Requirements (part 1)

Index	Requirement	[134]	[128]	[136]	[131, 132]	[208]	[123]	[170]	[101]	[3]	[126]	[125]	[100]	[148]	[234]	[187]	[213]	[193]	[176]	[89]	[88]	[243]	[34]	[162]
72	Process flexibility by change																	+						
73	Industry acceptance							+											+					
74	Semantic description to support dynamic service discovery and invocation																					+		
75	Measurements																							+
76	Stochastic modeling																							
77	Documentation				+	+	+	+																
78	Asynchronous and synchronous interaction	+	+	+	+	+		+														+		

Table A.1.: Sources of Requirements (part 1)

Index	Requirement	[161]	[47]	[9]	[13]	[87]	[199]
1	Multi-level and multi-view description	+		+	+	+	+
2	Support for business transactions						
3	Support for business signals						
4	Hierarchical decomposition; Composability		+	+	+	+	+
5	Support for binary collaborations				+		+
6	Support for multi-party collaborations						
7	Support for business documents			+	+	+	+
8	Quality of service			+			+
9	Control flow definition			+	+	+	+
10	Exception/Error handling			+		+	+
11	Role modeling	+			+		
12	Role mapping						
13	Support for business document attachments				+		
14	Support for process version control						
15	Technology independence of process model				+	+	+
16	Integration partner binding						
17	Flexible configuration of transfer/transport protocol						+
18	Flexible configuration of document exchange characteristics						
19	Negotiation of business capabilities						
20	Negotiation of communication capabilities						
21	Configuration data for runtime systems						
22	Interfacing with backend systems						+
23	Message correlation						

Table A.2.: Sources of Requirements (part 2)

Index	Requirement	[161]	[47]	[9]	[13]	[87]	[199]
24	Communication interface						
25	Registry functionality						
26	Repository functionality						+
27	Metadata definition					+	
28	Classification of processes						
29	Definition of associations between processes						
30	Cataloging of processes						
31	Validation			+			
32	Lifecycle management of B2Bi artifacts; Methodology						
33	Management of relationships among service/process providers and service/process users						
34	Process governance						
35	Extensibility						+
36	Event propagation						
37	Visual representation				+	+	+
38	Machine-processable format						
39	State-based modeling						
40	Data formats and data codes						
41	External communication						
42	Consistency						
43	Ease of maintenance						
44	Ease of explanation						
45	Auto-generation of artifacts						
46	Description of usage scenarios		+		+		
47	Description of business requirements						
48	Description of business benefits						
49	Pre/Post-conditions of process/task executions			+			
50	Formal methods		+	+			
51	Analysis features	+	+				
52	Intelligible feedback of analysis		+				

Table A.2.: Sources of Requirements (part 2)

Index	Requirement	[161]	[47]	[9]	[13]	[87]	[199]
53	Language for semantic constraint specification						
54	Semantic constraint management						
55	Semantic constraint violation traceability						
56	Reputation information management						
57	Language domain appropriateness			+			
58	Language participant knowledge appropriateness	+	+				
59	Language comprehensibility appropriateness	+	+	+			
60	Language technical actor appropriateness						
61	Language organizational appropriateness						
62	Reasonable tool support		+				
63	Traceability between process model and process execution			+			
64	Flexibility by underspecification						
65	Adaptability						
66	Dynamism						
67	Control flow patterns						
68	Data oriented process definition						
69	Simulation		+			+	
70	Usage of standards						
71	Process flexibility by design						
72	Process flexibility by change		+				
73	Industry acceptance						
74	Semantic description to support dynamic service discovery and invocation						
75	Measurements	+	+				
76	Stochastic modeling		+				
77	Documentation		+				

Table A.2.: Sources of Requirements (part 2)

Index	Requirement	[161]	[47]	[9]	[13]	[87]	[199]
78	Asynchronous and synchronous interaction			+			

Table A.2.: Sources of Requirements (part 2)

Index (ordered)	Index (survey)	Requirement	Requirement Group	Communication among unequal personnel	Agreement	Management of complex associations	Homogenization of computing resources	Comprehensibility	Feasibility	Changeability
1	58	Language participant knowledge appropriateness	Use specific media that help	2	0	0	0	1	0	0
2	52	Intelligible feedback of analysis	Use specific media that help	2	0	0	0	1	0	0
3	77	Documentation	Describe context of application	2	1	0	0	1	1	1
4	46	Description of usage scenarios	Describe context of application	2	0	0	0	1	0	0
5	47	Description of business requirements	Describe context of application	2	1	0	0	1	1	0
6	48	Description of business benefits	Describe context of application	2	0	0	0	1	1	0
7	28	Classification of processes	Group processes together	2	0	0	0	1	0	0
8	29	Definition of associations between processes	Group processes together	2	0	0	0	1	0	0
9	2	Support for business transactions	Define synchronization constructs	0	2	0	1	0	0	0
10	3	Support for business signals	Define synchronization constructs	0	2	0	1	0	0	0
11	5	Support for binary collaborations	Define synchronization constructs	0	2	0	0	1	0	0
12	7	Support for business documents	Define synchronization constructs	0	2	0	0	0	0	0
13	9	Control flow definition	Define synchronization constructs	0	2	0	1	0	2	1
14	53	Language for semantic constraint specification	Describe state space of collaboration	1	2	0	0	1	1	1
15	68	Data oriented process definition	Describe state space of collaboration	1	2	0	0	0	0	0
16	49	Pre/Post-conditions of process/task executions	Describe state space of collaboration	1	2	0	0	0	1	0
17	16	Integration partner binding	Allow for partner specifics	0	2	0	0	0	0	0
18	19	Negotiation of business capabilities	Allow for partner specifics	0	2	1	0	0	0	0
19	20	Negotiation of communication capabilities	Allow for partner specifics	0	2	1	1	0	0	0
20	56	Reputation information management	Allow for unknown partners	0	0	2	0	0	0	0
21	6	Support for multi-party collaborations	Define constructs for complex interactions	0	1	2	0	0	0	0
22	11	Role modeling	Define constructs for complex interactions	1	0	2	0	1	0	1
23	12	Role mapping	Define constructs for complex interactions	0	0	2	0	0	0	1
24	67	Control flow patterns	Define constructs for complex interactions	1	1	2	1	1	1	0
25	17	Flexible configuration of transfer/transport protocol	Overcome technical communication obstacles	0	0	2	1	0	0	1
26	27	Metadata definition	Manage associations	0	0	2	1	0	0	1
27	30	Cataloging of processes	Manage associations	0	0	2	0	0	0	1
28	36	Event propagation	Manage associations	0	0	2	0	0	0	0
29	74	Semantic description to support dynamic service discovery and invocation	Manage associations	0	1	2	0	0	0	1

Table A.3.: Requirements-Challenge Relation

Index (ordered)	Index (survey)	Requirement	Requirement Group	Communication among unequal personnel	Agreement	Management of complex associations	Homogenization of computing resources	Comprehensibility	Feasibility	Changeability
30	25	Registry functionality	Manage associations	0	0	2	0	0	0	1
31	26	Repository functionality	Manage associations	0	0	2	0	0	0	1
32	54	Semantic constraint management	Use association management facilitators	0	1	2	1	0	1	1
33	14	Support for process version control	Use association management facilitators	0	1	2	0	0	0	1
34	70	Usage of standards	Use association management facilitators	1	1	2	1	0	1	0
35	40	Data formats and data codes	Use association management facilitators	1	1	2	1	0	0	0
36	43	Ease of maintenance	Use association management facilitators	0	0	2	1	0	1	1
37	78	Asynchronous and synchronous interaction	Deal with basic distributed interaction styles	0	0	1	2	0	1	0
38	8	Quality of service	Deal with distributed communication	0	1	0	2	0	0	0
39	10	Exception/Error handling	Deal with distributed communication	0	1	0	2	0	0	0
40	23	Message correlation	Deal with distributed communication	0	0	1	2	0	0	0
41	24	Communication interface	Deal with distributed communication	0	0	0	2	0	1	0
42	42	Consistency	Manage state space	0	1	0	2	0	1	0
43	55	Semantic constraint violation traceability	Manage state space	0	0	1	2	0	0	0
44	21	Configuration data for runtime systems	Deal with heterogeneity	0	1	0	2	0	0	1
45	22	Interfacing with backend systems	Deal with heterogeneity	0	0	1	2	0	1	1
46	1	Multi-level and multi-view description	Decompose problem	1	0	0	0	2	0	0
47	4	Hierarchical decomposition; Composability	Decompose problem	1	0	0	1	2	0	0
48	37	Visual representation	Use adequate representations	1	0	0	0	2	0	0
49	39	State-based modeling	Use adequate representations	1	1	0	1	2	0	0
50	44	Ease of explanation	Use adequate representations	1	0	0	0	2	0	0
51	59	Language comprehensibility appropriateness	Use adequate representations	0	0	0	0	2	0	0
52	69	Simulation	Figure out what's going on	0	0	1	0	2	1	1
53	75	Measurements	Figure out what's going on	1	1	0	1	2	0	0
54	76	Stochastic modeling	Figure out what's going on	1	0	1	1	2	1	0
55	13	Support for business document attachments	Adapt to real world as is	0	1	0	0	0	2	0
56	18	Flexible configuration of document exchange characteristics	Adapt to real world as is	0	0	1	0	0	2	1
57	41	External communication	Adapt to real world as is	0	1	0	0	0	2	0
58	64	Flexibility by underspecification	Adapt to real world as is	0	0	1	0	0	2	0
59	65	Adaptability	Adapt to real world as is	0	0	1	1	0	2	0
60	71	Process flexibility by design	Adapt to real world as is	0	0	1	1	0	2	0
61	72	Process flexibility by change	Adapt to real world as is	0	0	1	1	0	2	1
62	73	Industry acceptance	Offer wanted/needed functionality	1	1	1	1	0	2	0

Table A.3.: Requirements-Challenge Relation

Index (ordered)	Index (survey)	Requirement	Requirement Group	Communication among unequal personnel	Agreement	Management of complex associations	Homogenization of computing resources	Comprehensibility	Feasibility	Changeability
63	57	Language domain appropriateness	Offer wanted/needed functionality	1	0	0	0	1	2	0
64	62	Reasonable tool support	Offer wanted/needed functionality	0	1	1	1	1	2	1
65	31	Validation	Realize business alignment	0	0	1	1	0	2	0
66	32	Lifecycle management of B2Bi artifacts; Methodology	Realize business alignment	0	0	1	1	0	2	1
67	34	Process governance	Realize business alignment	0	0	1	0	0	2	1
68	61	Language organizational appropriateness	Realize business alignment	0	0	0	0	0	2	0
69	38	Machine-processable format	Use automation	0	1	1	0	0	2	1
70	45	Auto-generation of artifacts	Use automation	0	1	1	1	0	2	1
71	51	Analysis features	Use automation	0	1	1	0	1	2	1
72	60	Language technical actor appropriateness	Use automation	0	1	1	1	0	2	1
73	63	Traceability between process model and process execution	Use automation	0	1	1	1	0	2	0
74	15	Technology independence of process model	Use abstraction	1	0	1	1	1	0	2
75	33	Management of relationships among service/process providers and service/process users	Manage associations	0	0	1	0	0	0	2
76	35	Extensibility	Allow for evolution	0	0	1	1	0	1	2
77	66	Dynamism	Allow for evolution	0	0	1	0	0	1	2
78	50	Formal methods	Use formal methods	1	1	1	1	1	1	1

Table A.3.: Requirements-Challenge Relation

Index (ordered)	Index (survey)	Requirement	Requirement Group	Business Model	Business Process Model	Choreography Model	Public Orchestration Definition	Private Orchestration Definition	Runtime Systems	Sum
1	58	Language participant knowledge appropriateness	Use specific media that help	1	2	1	0	0	0	4
2	52	Intelligible feedback of analysis	Use specific media that help	1	2	1	1	1	2	8
3	77	Documentation	Describe context of application	2	2	2	1	1	2	10
4	46	Description of usage scenarios	Describe context of application	2	2	1	0	1	0	6
5	47	Description of business requirements	Describe context of application	2	2	0	0	0	0	4
6	48	Description of business benefits	Describe context of application	2	1	0	-	-	0	3

Table A.4.: Requirements-Abstraction Layer Relation

Index (ordered)	Index (survey)	Requirement	Requirement Group	Business Model	Business Process Model	Choreography Model	Public Orchestration Definition	Private Orchestration Definition	Runtime Systems	Sum
7	28	Classification of processes	Group processes together	1	2	1	0	0	1	5
8	29	Definition of associations between processes	Group processes together	2	2	1	1	2	2	10
9	2	Support for business transactions	Define synchronization constructs	-	1	2	2	2	1	8
10	3	Support for business signals	Define synchronization constructs	-	0	1	2	2	1	6
11	5	Support for binary collaborations	Define synchronization constructs	-	2	2	2	1	0	7
12	7	Support for business documents	Define synchronization constructs	0	2	2	2	2	2	10
13	9	Control flow definition	Define synchronization constructs	-	2	1	2	2	0	7
14	53	Language for semantic constraint specification	Describe state space of collaboration	1	2	2	1	1	1	8
15	68	Data oriented process definition	Describe state space of collaboration	0	2	2	1	2	0	7
16	49	Pre/Post-conditions of process/task executions	Describe state space of collaboration	1	2	2	0	0	1	6
17	16	Integration partner binding	Allow for partner specifics	-	0	1	1	2	2	6
18	19	Negotiation of business capabilities	Allow for partner specifics	2	2	1	1	0	-	6
19	20	Negotiation of communication capabilities	Allow for partner specifics	0	1	2	2	0	0	5
20	56	Reputation information management	Allow for unknown partners	1	1	1	1	1	1	6
21	6	Support for multi-party collaborations	Define constructs for complex interactions	0	2	2	2	1	0	7
22	11	Role modeling	Define constructs for complex interactions	1	2	2	1	0	0	6
23	12	Role mapping	Define constructs for complex interactions	0	2	2	2	1	0	7
24	67	Control flow patterns	Define constructs for complex interactions	-	2	1	2	2	0	7
25	17	Flexible configuration of transfer/transport protocol	Overcome technical communication obstacles	-	-	1	2	0	1	4
26	27	Metadata definition	Manage associations	0	1	2	2	1	0	6
27	30	Cataloging of processes	Manage associations	-	1	1	1	1	2	6
28	36	Event propagation	Manage associations	-	1	1	1	1	2	6
29	74	Semantic description to support dynamic service discovery and invocation	Manage associations	-	1	2	2	1	2	8
30	25	Registry functionality	Manage associations	0	1	2	2	1	1	7
31	26	Repository functionality	Manage associations	-	0	1	1	1	2	5
32	54	Semantic constraint management	Use association management facilitators	0	1	2	2	1	1	7
33	14	Support for process version control	Use association management facilitators	0	2	2	2	2	1	9
34	70	Usage of standards	Use association management facilitators	1	1	2	2	1	1	8
35	40	Data formats and data codes	Use association management facilitators	-	0	1	2	2	1	6
36	43	Ease of maintenance	Use association management facilitators	-	0	0	1	2	1	4

Table A.4.: Requirements-Abstraction Layer Relation

Index (ordered)	Index (survey)	Requirement	Requirement Group	Business Model	Business Process Model	Choreography Model	Public Orchestration Definition	Private Orchestration Definition	Runtime Systems	Sum
37	78	Asynchronous and synchronous interaction	Deal with basic distributed interaction styles	-	1	1	2	2	1	7
38	8	Quality of service	Deal with distributed communication	0	0	1	2	2	2	7
39	10	Exception/Error handling	Deal with distributed communication	-	1	1	1	2	2	7
40	23	Message correlation	Deal with distributed communication	-	0	1	2	1	1	5
41	24	Communication interface	Deal with distributed communication	-	0	0	1	1	2	4
42	42	Consistency	Manage state space	1	2	1	2	2	1	9
43	55	Semantic constraint violation traceability	Manage state space	0	1	1	1	1	2	6
44	21	Configuration data for runtime systems	Deal with heterogeneity	-	-	0	1	1	2	4
45	22	Interfacing with backend systems	Deal with heterogeneity	-	0	0	0	2	1	3
46	1	Multi-level and multi-view description	Decompose problem	1	2	1	1	1	0	6
47	4	Hierarchical decomposition; Composability	Decompose problem	0	2	2	2	2	0	8
48	37	Visual representation	Use adequate representations	1	2	1	1	1	0	6
49	39	State-based modeling	Use adequate representations	1	2	1	1	1	0	6
50	44	Ease of explanation	Use adequate representations	1	2	1	1	1	1	7
51	59	Language comprehensibility appropriateness	Use adequate representations	2	2	1	1	1	-	7
52	69	Simulation	Figure out what's going on	1	2	1	2	2	0	8
53	75	Measurements	Figure out what's going on	1	1	0	0	1	2	5
54	76	Stochastic modeling	Figure out what's going on	1	2	0	0	1	1	5
55	13	Support for business document attachments	Adapt to real world as is	-	0	1	2	1	1	5
56	18	Flexible configuration of document exchange characteristics	Adapt to real world as is	-	0	2	2	1	0	5
57	41	External communication	Adapt to real world as is	-	1	1	1	2	1	6
58	64	Flexibility by underspecification	Adapt to real world as is	0	2	1	1	2	1	7
59	65	Adaptability	Adapt to real world as is	0	2	1	1	2	1	7
60	71	Process flexibility by design	Adapt to real world as is	1	2	1	1	2	0	7
61	72	Process flexibility by change	Adapt to real world as is	1	1	1	1	2	1	7
62	73	Industry acceptance	Offer wanted/needed functionality	1	1	1	1	0	0	4
63	57	Language domain appropriateness	Offer wanted/needed functionality	1	2	2	1	1	-	7
64	62	Reasonable tool support	Offer wanted/needed functionality	1	2	2	2	2	1	10
65	31	Validation	Realize business alignment	1	1	2	2	2	0	8
66	32	Lifecycle management of B2Bi artifacts; Methodology	Realize business alignment	0	1	2	2	1	0	6
67	34	Process governance	Realize business alignment	1	2	2	2	2	0	9
68	61	Language organizational appropriateness	Realize business alignment	1	2	2	1	1	-	7
69	38	Machine-processable format	Use automation	0	1	2	2	2	2	9

Table A.4.: Requirements-Abstraction Layer Relation

Index (ordered)	Index (survey)	Requirement	Requirement Group	Business Model	Business Process Model	Choreography Model	Public Orchestration Definition	Private Orchestration Definition	Runtime Systems	Sum
70	45	Auto-generation of artifacts	Use automation	-	0	1	2	2	1	6
71	51	Analysis features	Use automation	1	2	2	2	2	2	11
72	60	Language technical actor appropriateness	Use automation	0	1	2	2	2	1	8
73	63	Traceability between process model and process execution	Use automation	0	1	1	1	2	2	7
74	15	Technology independence of process model	Use abstraction	%	2	1	0	0	-	3
75	33	Management of relationships among service/process providers and service/process users	Manage associations	0	0	1	2	2	1	6
76	35	Extensibility	Allow for evolution	1	1	1	2	2	2	9
77	66	Dynamism	Allow for evolution	-	0	0	1	2	2	5
78	50	Formal methods	Use formal methods	0	2	2	2	1	0	7
			Sum:	40	100	98	104	101	66	

Table A.4.: Requirements-Abstraction Layer Relation

B. Algorithm for Translating ebBP-ST to BPEL

This appendix contains research results that have been created by Christoph Pflügler for his diploma thesis which has been supervised during this work's dissertation project. The contents have been adapted from [182].

A WSTBC is translated into two orchestrated BPEL processes using a two-stage procedure. At first, the control flow of a WSTBC is translated, whereas a placeholder for each `BTA` with its subsequent `Decision` is inserted. Then, each of these placeholders is replaced with the respective BPEL code. The overall procedure is depicted in algorithm 11 and described in more detail in the following paragraphs. Thereby, each *write** function takes the reference to the file containing the BPEL process of a role r∈R as first parameter. All other parameters are function specific. Note that each of the *write** functions appends its BPEL code to the end of the BPEL process definition file specified in the first parameter.

The translation algorithm begins with the insertion of the BPEL code required before being able to insert the BPEL translation of the actual control flow. This code includes the import of required WSDL interfaces, the specification of `partnerLinks`, the declaration of global `variables` such as the ones containing the process state, as well as the specification of a business document independent `correlationSet`. Furthermore, `collaboration timeouts` are handled and the so-called *internal process state* is initialized with the `nameID` of the shared state which can be reached from the initial state (`nameID` of $parentST(out(wbc.s_0))$). Finally, the central `while` loop switching over all shared states is prepared. Listing B.1 shows a corresponding BPEL snippet.

Listing B.1: BPEL Output of writeBPELHeader Function

```
<process ... name="UseCase">
 <!-- WSDL imports here -->
 <!-- parterLinks here -->
 <!-- variables here -->
 <!-- correlation set here -->
  <scope name="UseCase">
 <eventHandlers>
  <onAlarm>
   <!-- collaboration timeout handling -->
  </onAlarm>
 </eventHandlers>
 <sequence>
  <assign>
   <copy>
    <from>
     <literal>
      <wsdlDoc:stateType>{first state nameID}
```

Algorithm 11: WSTBC to BPEL Translation Algorithm

```
input     : A valid WSTBC $wbc$ to be transformed
output    : Two orchestrated BPEL processes $BPELprocesses<wbc.r_1$.bpel,
            $wbc.r_2$.bpel>
algorithm :
1  $BPELprocesses<$writeBPELHeader($wbc.r_1.bpel,$ parentST( out($wbc.s_0$))),
   writeBPELHeader($wbc.r_2.bpel,$ parentST( out($wbc.s_0$)))$>$;
2  $BPELprocesses<$writeBPELStateProlog($wbc.r_1.bpel,$ parentST(out($wbc.s_0$)),
   $tt$), writeBPELStateProlog($wbc.r_2.bpel,$ parentST( out($wbc.s_0$)), $tt$)$>$;
3  foreach $link$ in out(nodef( parentST( out($wbc.s_0$)))) do
4      $BPELprocesses<$writeBTA+DECplaceholder($wbc.r_1.bpel, link,$ out($link$)),
       writeBTA+DECplaceholder($wbc.r_2.bpel, link,$ out($link$))$>$;
5  end
6  $BPELprocesses<$writeBPELStateEpilog($wbc.r_1.bpel$),
   writeBPELStateEpilog($wbc.r_2.bpel$)$>$;
7  foreach $st$ in $wbc.ST$ | $st \neq$ parentST( out($s_0$)) do
8      $BPELprocesses<$writeBPELStateProlog($wbc.r_1.bpel, st, ff$),
       writeBPELStateProlog($wbc.r_2.bpel, st, ff$)$>$;
9      foreach $link$ in out(nodef($st$)) do
10         $BPELprocesses<$writeBTA+DECplaceholder($wbc.r_1.bpel, link,$
           out($link$)), writeBTA+DECplaceholder($wbc.r_2.bpel, link,$ out($link$))$>$;
11     end
12     $BPELprocesses<$writeBPELStateEpilog($wbc.r_1$).bpel,
       writeBPELStateEpilog($wbc.r_2.bpel$)$>$;
13 end
14 foreach $t$ in $wbc.T$ |$\exists(n_k, tt, t)\in ctrlFlow(wbc) \wedge \exists(n_l, g, n_k)\in ctrlFlow(wbc)$ do
15     $BPELprocesses<$writeBPELTerminalCode($wbc.r_1.bpel, t$),
       writeBPELTerminalCode($wbc.r_2.bpel, t$)$>$;
16 end
17 $BPELprocesses<$writeBPELProcessEpilog($wbc.r_1.bpel$),
   writeBPELProcessEpilog($wbc.r_2.bpel$)$>$;
18 $BPELprocesses<$replaceBTA+DECplaceholders($wbc.r_1.bpel$),
   replaceBTA+DECplaceholders($wbc.r_2.bpel$)$>$;
19 return $BPELprocesses<wbc.r_1$.bpel, $wbc.r_2$.bpel>;
```

```
        </wsdlDoc:stateType>
      </literal>
     </from>
     <to>$processState_internal</to>
    </copy>
   </assign>
   <while>
    <condition>'true'</condition>
    <sequence>
```

After initializing the two BPEL processes, the shared state which can be reached from the initial state is translated first before translating all remaining ones. The only difference between the translation of the first and the remaining shared states is that the execution of one of the BTAs admissible from the first shared state creates a new process instance (`createInstance="yes"`). The translation of a shared state is carried out in three steps: At first, a so-called state prolog providing the functionality for state timeout handling is added (listing B.2). Thereafter, the placeholders for all `BTA`s with attached `Decision`s (cf. listing B.3) admissible from the respective shared state are inserted. Finally, a so-called state epilog (c.f. listing B.4) closes all remaining open tags related to the shared state BPEL code.

Listing B.2: BPEL Output of writeBPELStateProlog Function

```
<if>
 <condition>$processState_internal = '{state nameID}'</condition>
 <scope name="{state nameID}_Timeout_Scope">
  <faultHandlers>
   <catch faultName="StateTimeout">
    <empty />
   </catch>
  </faultHandlers>
  <scope name="{state nameID}_Scope">
   <eventHandlers>
    <onAlarm>
     <for>'{state ttp}'</for>
     <scope>
      <sequence>
       <assign>
        <copy>
         <from>
          <literal>
           <wsdlDoc:stateType>{nameID of state reached in case of state timeout}</
               wsdlDoc:stateType>
          </literal>
         </from>
         <to>$processState_internal</to>
        </copy>
       </assign>
       <throw faultName="StateTimeout" />
      </sequence>
     </scope>
    </onAlarm>
   </eventHandlers>
   <sequence>
    <assign>
     <copy>
      <from>
       <literal>
        <wsdlDoc:stateType>{state inner entry nameID}</wsdlDoc:stateType>
       </literal>
      </from>
      <to>$processState_internal</to>
     </copy>
    </assign>
    <while>
     <condition>$processState_internal =
      '{state inner entry nameID}'
```

```
        </condition>
      <sequence>
       <assign>
        <copy>
         <from>
          <literal>
           <wsdlDoc:stateType>{state nameID}
            </wsdlDoc:stateType>
          </literal>
         </from>
         <to>$processState</to>
        </copy>
       </assign>
       <invoke operation="dropProcessState" .. inputVariable="processState" />
       <!-- createInstance="yes", if second parameter of function is true,
           createInstance="no" otherwise -->
       <pick createInstance="yes">
```

In order to implement the shared state timeout behaviour described in section 4.4 in a BPEL standard compliant manner, two distinct variables for the process state are necessary. The `processState_internal` variable is used for internal purposes only and governs the execution of the BPEL processes according to the ebBP process definition. The second one, `processState`, is used to communicate the state of a collaboration instance to the collaboration partners. Note that the state is assigned to `processState` only after a BPEL process already is in the respective state. Besides the `nameID`s of all shared states of a business collaboration which constitute all possible values of `processState`, `processState_internal` also includes the `nameID`s of all timer-preserving $nodei(st)$ (cf. section 4.4) of all shared states st as possible values. It is important to know that in BPEL, a `scope` in which a `fault` occurred is considered to have completed unsuccessfully [137]. Throwing a `fault` terminates all `scopes` this `fault` is thrown in or passed through until it is handled in some `scope`. Hence, if a state timeout is reached, the `nameID` of the state specified in the respective timeout `linkTo` is assigned to `processState_internal` and a fault terminating all `scopes` not handling it is thrown. The outmost scope of the BPEL code for a shared state handles the fault and subsequently also terminates. Thus, the process switches to the shared state specified in the timeout `linkTo`. If a `Decision` attached to an admissible `BTA` of a shared state links to $nodei(st)$ of that shared state st, the process remains in that state without resetting the timer due to the `while`-loop. Conversely, if that `Decision` links to $nodej(st)$ of the shared state st, the next iteration of the `while`-loop switching over all shared states is triggered, and the timer of the shared state is reset consequently.

Listing B.3: BPEL Output of writeBTA+DECplaceholder Function

```
<empty name="{BTA nameID}###{DECISION NameID}" />
```

Once the state prolog for a state is appended, a placeholder for each `BTA` with attached `Decision` admissible from the respective shared state is added. These placeholders are replaced by BPEL code later in the translation procedure. The state epilog finally closes all open BPEL tags corresponding to the translation of a shared state.

Listing B.4: BPEL Output of writeBPELStateEpilog Function

```
       </pick>
      </sequence>
    </while>
   </sequence>
  </scope>
 </scope>
</if>
```

After translating all shared states, all terminal states need to be translated as well. Thereby, the BPEL code depicted in listing B.5 is inserted for each terminal state referenced by a `Decision` or a timeout `linkTo` of a shared state. As in every shared state, the process state is pushed to the backend system using the `invoke` construct.

Listing B.5: BPEL Output of writeBPELTerminalCode Function

```
<if>
 <condition>$processState_internal = '{terminal state nameID}'</condition>
 <sequence>
  <assign>
   <copy>
    <from>
     <literal>
      <wsdlDoc:stateType>{terminal state nameID}<wsdlDoc:stateType>
     </literal>
    </from>
    <to>$processState</to>
   </copy>
  </assign>
  <invoke operation="dropProcessState" .. inputVariable="processState" />
  <exit/>
 </sequence>
</if>
```

In the next step, all open BPEL tags of the entire process (c.f. listing B.6) need to be closed.

Listing B.6: BPEL Output of writeBPELProcessEpilog Function

```
    </sequence>
   </while>
  </sequence>
 </scope>
</process>
```

Before ending the translation procedure, all placeholders for `BTA`s and attached `Decision`s need to be replaced by the respective BPEL code. As this appendix focuses on the introduction of shared states in modeling ebBP business collaborations, the translation of `BTA`s and `Decision`s is depicted more briefly in the rest of this section.

Tables B.1, B.2/B.3 and B.4 give an overview of the most important BPEL elements used to translate a BTA with an attached Decision and indicate the purpose of their particular usage. The elements are listed in the order of their occurrence in the BPEL `process`. The tables' content corresponds to a BTA that contains a `ReceiptAcknowledgement` as well as an `AcceptanceAcknowledgement` signal. Furthermore, all timing parameters offered for a BTA by the ebBP specification [134] are set. If only some or none of the signals for and parameters of a BTA are specified, the BPEL translation is a corresponding subset of the depicted one. As the mapping of a `RespondingBusinessActivity` is analogous to a `Requesting-BusinessActivity`, only a `RequestingBusinessActivity` is described here. An

Table B.1.: BPEL Production Rules for ebBP BusinessTransactionActivity

Role	BPEL Process Elements
Initiator + Responder	- enclosing `onMessage`, receiving a triggering message from integration partner or backend system - enclosing `scope` for complete BTA - all `variable`s required for the ebBP *Requesting-/RespondingBA* and the ebBP *Decision* - `catch` blocks for all ebBP failure types containing the corresponding reaction as specified in the ebBP *BusinessCollaboration* - `catchAll` block containing reaction as specified in ebBP *BusinessCollaboration* for *AnyProtocolFailure* - `onAlarm` to implement the *TimeToPerform* parameter specified for the BTA - `sequence` containing the BPEL code for the ebBP constructs (in this order): *RequestingBA*, *RespondingBA*, *Decision.* For the respective production rules see tables B.2/B.3 and B.4.

occurrence of an `onAlarm` based timeout in combination with throwing a `fault` terminates the BTA and is handled by the `faultHandlers` of the `scope` enclosing the BPEL code of a BTA. `ReceiptAcknowledgement/-Exception` (RA/RAE) and `AcceptanceAcknowledgement/-Exception` (AA/AAE) are processed concurrently as suggested by the ebBP specification [134, sec. 3.4.9.3.3]. A process tries to get a valid *RA/RAE* by sending the corresponding business document (BD) to the process of the integration partner until the specified ebBP *retryCount* is exceeded or a timeout occurs. Furthermore, if both signal types are used, it waits for receiving a valid *AA/AAE* until the occurrence of a timeout. At the end of the BTA mapping, an ebBP *Decision* is realized by using an `invoke` for querying the backend services for the evaluation of the latest business document exchanged and the `processState_internal` variable is then set correspondingly. As described earlier, this may lead to a switch to another shared state within the next iteration of the collaboration's `while` loop.

Table B.2.: BPEL Production Rules for ebBP RequestingBusinessActivity (1)

Role	BPEL Process Elements
Initiator + Responder	- enclosing `scope` - enclosing `flow` for concurrent processing of *RA/RAE* and *AA/AAE*
	RA / RAE
Initiator	- enclosing `while` for trying to get a valid *RA/RAE* until ebBP *retryCount* is exceeded - `scope` to encapsulate *RA/RAE* handling - `catch` block for *RAE* handling - `invoke` to check *RAE* validity using the backend system - `throw` to throw ebBP *AnyProtocolFailure* in case no valid *RAE* was received and ebBP *retryCount* is exceeded - `throw` to throw ebBP *RequestReceiptFailure* in case of a valid *RAE* - `catchAll` block for technical failure (TF) handling - `rethrow` to forward TF to enclosing `scope` if ebBP *retryCount* is exceeded - `onAlarm` to implement ebBP *timeToAcknowledgeReceipt* - `throw` ebBP *AnyProtocolFailure* if ebBP *retryCount* is exceeded - `invoke` to forward Business Document (BD) to and get a *RA/RAE* from Responder - `invoke` to check *RA* validity using the backend system
Responder	- enclosing `scope` - `catch` block for *RAE* handling - `reply` construct to forward *RAE* to Initiator - `throw` ebBP *RequestReceiptFailure* in case of a *RAE* - `onAlarm` to implement ebBP *timeToAcknowledgeReceipt* - `invoke` to forward BD to and get a *RA/RAE* from backend system - `reply` to forward *RA* to Initiator

Table B.3.: BPEL Production Rules for ebBP RequestingBusinessActivity (2)

Role	BPEL Process Elements
	AA / AAE
Initiator	- enclosing `while` to wait for valid *AA/AAE* - `scope` to encapsulate *AA/AAE* handling - `catch` block to forward ebBP *RequestAcceptanceFailure* faults to enclosing `scope`s - `catchAll` block to handle TF - `empty` to wait for an *AA/AAE* despite of TFs - `onAlarm` to implement ebBP *timeToAcknowledgeAcceptance* - `pick` to receive either *AA* or *AAE* - `invoke` to check *AA/AAE* validity using the backend system - `throw` ebBP *RequestAcceptanceFailure* in case of a valid *AAE*
Responder	- enclosing `scope` - `catch` block for handling *AAE* - `invoke` to forward *AAE* to Initiator - `throw` ebBP *RequestAcceptanceFailure* in case of an *AAE* - `onAlarm` to implement ebBP *timeToAcknowledgeAcceptance* - `invoke` to get *AA/AAE* from backend system - `invoke` to forward *AA* to Initiator

Table B.4.: BPEL Production Rules for ebBP Decision

Role	BPEL Process Elements
Initiator + Responder	- `invoke` to send BD of *RespondingBA* to backend system in order to get an evaluation - if no *RespondingBA* exists, BD of *RequestingBA* is used - `if` and `assign` statements to determine and switch to next process state Note that *ConditionGuardValues* are evaluated before *DocumentEnvelopes*.

C. WS-* Implementation of the Secure WS-ReliableMessaging Scenario

This appendix contains research results that have been created by Johannes Schwalb for his diploma thesis which has been supervised during this work's dissertation project. The contents have been adapted from [196].

The purpose of this appendix is the analysis of the availability of the SecRM scenario on contemporary WS stacks. Such an investigation is desirable even if interoperable WS-* functionality is not available (cf. [183, 196]) because there are use cases of WS-RM and WS-Sec that are desirable for homogeneous environments as well. Content-level encryption and signing of XML messages allows for multi-party scenarios that barely can be matched using transport level security methods. The combination with additional WS-* standards such as WS-Trust [146] and WS-SecureConversation [144] allows integration scenarios that obviate the need of mutual security certificate exchanges outside of the actual Web Services interaction.

The purpose of the SecRM scenario is the reliable exchange of confidential, authenticated and integrity-protected messages without frequent key exchanges (cf. [4, 19] for more detailed descriptions of security properties). In order to achieve these goals WS-Sec, WS-RM, WS-Trust, WS-SecureConversation as well as WS-Addressing are combined for establishing a WS-SecureConversation *session* (or *security context*) which is then used to reliably and *securely* exchange messages. Basically, the SecRM scenario consists of a key-exchange phase, a message sending phase and a termination phase [4]. The key-exchange phase generates a so-called Security Context Token (SCT) and uses asymmetric keys for integrity as well as confidentiality protection of WS-SecureConversation bootstrap messages. After the generation of the security context, a WS-RM sequence is initiated which is used for exchanging payload messages. Once the exchange of all payload-messages has been acknowledged, the WS-RM sequence as well as the security context are subsequently terminated.

The steps listed below describe the interactions between the client and the sender role of the SecRM scenario and reflect the definitions of [4] and [44, pages 34-60]. The prefixes `wsrm`, `wss`, `wst`, `wsu` and `env` are used to identify concepts of the WS-RM, WS-Sec, WS-Trust, WS-Sec utility and SOAP standards, respectively.

1. **RequestSecurityToken RST:** The client sends a message containing a RST to the service, asking the service to issue a SCT. This message must be signed and encrypted by the client using the service's public key.

2. **RequestSecurityTokenResponse RSTR:** The service responds with a RSTR message, containing the requested SCT. This message must be signed and encrypted by the service using the client's public key. The SCT must be used for signing and encrypting any message of the subsequent message flow.

3. **CreateSequence:** The client sends a `wsrm:CreateSequence` message to the service. This message includes a `wss:SecurityContextReference` to reference the SCT received in step 2. This SCT is used to sign the `wsrm:CreateSequence` message and encrypt the `wss:Signature`.

4. **CreateSequenceResponse:** The service responds to the CreateSequence request with a `wsrm:CreateSequenceResponse` message. This message is also signed and the `wss:Signature` is encrypted using the SCT.

5. **Payload Message:** The client now sends signed and encrypted messages containing the payload of this communication. Each payload message contains a WS-ReliableMessaging sequence header containing at least the sequence identifier and the sequence number of the respective message. In contrast to the previous two and following five messages, the payload messages have an encrypted `env:Body`.

6. **SequenceAcknowledgement:** The service acknowledges the receipt of the payload message(s) with a `wsrm:SequenceAcknowledgement`. The scenario definition proposes a single `wsrm:SequenceAcknowledgement` message with an empty `env:Body`. The acknowledgment headers are also signed and the `wss:Signature` is encrypted.

7. **TerminateSequence:** As soon as the client has received the acknowledgments for each message within the message sequence, it closes this sequence using the `wsrm:TerminateSequence` message defined by WS-RM. This message is signed and the `wss:Signature` is encrypted, too.

8. **TerminateSequenceResponse:** The service confirms the termination of the sequence with a signed `wsrm:TerminateSequenceResponse` message. The `wss:Signature` of this message is encrypted.

9. **CancelSecurityToken:** After termination of the WS-ReliableMessaging sequence, the client asks the service for cancellation of the WS-SecureConversation context using a `wst:CancelTarget` message. The WS-Addressing header elements, the `wss:Signature`, and the `wsu:Timestamp` are integrity protected. In addition, the signature is encrypted whereas the message `env:Body` is not protected.

10. **CancelSecurityTokenResponse:** The service confirms the `wst:CancelTarg et` with a `wst:RequestedTokenCancelled` message. This message is protected in the same way as the `wst:CancelTarget` message.

Once the security context is started, sender and receiver may create multiple WS-RM sessions for message transmission, as the publications (cf. [4, 19, 44]) do not impose any restrictions on that. However, in order to fulfill the requirements of the SecRM scenario each WS-RM session that is started within the WS-SecureConversation session must be closed or terminated within the same session. After the security context is established, all signature and encryption processes are performed using keys derived from the SCT. The last two messages that collectively cancel the security context are protected using SCT-derived keys, too.

Note that [4] and [44] make use of WS-ReliableMessaging version 1.0 [12] which does not define `wsrm:CloseSequence`, `wsrm:CloseSequenceResponse`, or `wsrm:TerminateSequenceResponse` messages. However, these messages have been defined for WS-RM since version 1.1 [138]. Therefore, message number eight `wsrm:TerminateSequenceResponse` is inserted into the message sequence as recommended by WS-I'S RSP [240, pages 28/29].

Table C.1 lists the requirements of the scenario definitions for confidentiality and integrity protection. The corresponding message type number is put into the first column. The other columns show the key required for encrypting (enc) or signing (sig) the message elements `wss:Signature` (Sign), `env:Body`, `wsu:Timestamp` (TS), WS-Addressing headers (WS-A), WS-ReliableMessaging headers (WS-R), and `wss:EncryptedKey` (EncKey). The entry SK_X stands for a session key, which is usually encrypted using the receivers public key (PuK_Y). The private key of a party is abbreviated PrK_Y. If a `wsc:SecurityContextToken` is used to derive keys, these keys are labeled as DK_{YX}. The indices X and Y are variables that are substituted in the table. Instead of the X a number is inserted to identify different instances of the corresponding key type. These instances are independent of the type of party. The index Y determines whether the client (c) or the service (s) is the owner of the key, e.g., PrK_S denotes that the key used for the specified operation is the private key of the service, while DK_{C1} stands for a key derived from a `SecurityContextToken` by the client. Since multiple derived keys may be used in a SOAP message, each derived key has an assigned number, here '1'. The '•' symbol indicates that the corresponding element is present, but not protected, whereas the '◦' means that the corresponding element is not present in this message.

The defining sources of the SecRM scenario describe the messages to be exchanged in detail. However, no WS-Policy definitions are given to instruct the WS stacks under test to create the messages as intended. Therefore the policy configuration of the SecRM scenario is sketched in section C.1. These assertions have been derived from the scenario descriptions and the sample messages provided in [44]. Section C.2 then checks the generated message exchanges by the GlassFish-openESB and IBM Websphere platforms following the approach of [197].

Msg	**Sign**	**Body**		**TS**	**WS-A**	**WS-R**	**EncKey**
No	**enc**	**enc**	**sig**	**sig**	**sig**	**sig**	**enc**
1	SK_1		PrK_C			○	PuK_S
2	SK_2		PrK_S			○	PuK_C
3	DK_{C1}	●	DK_{C2}			○	○
4	DK_{S1}	●	DK_{S2}			○	○
5	DK_{C1}		DK_{C2}				○
6	DK_{S1}	●	●	DK_{S2}			○
7	DK_{C1}	●	DK_{C2}				○
8	DK_{S1}	●	DK_{S2}				○
9	DK_{C1}	●	DK_{C2}			○	○
10	DK_{S1}	●	DK_{S2}			○	○

Table C.1.: Message Protection Requirements and Keys Required for Protection Realization Defined by the Scenario Definitions

C.1. Policy Configuration

The policy configurations below sketch the platform-independent specification of the service's WSDL definition for the SecRM scenario. In order to deploy these configurations on the selected platforms, some specific assertions have to be defined. For example, XPath-based `Encrypted`- and `SignedElements` assertions (cf. [145]) have to be added to work around IBM WebSphere's non-compliant handling of `EncryptSignature`, `EncryptBeforeSigning` or `Timestamp` assertions. For such details, please see [196].

The presentation of the WS-Policy definition is split up into four listings where listing C.1 shows the security binding and WS-RM configuration for the actual payload messages, listing C.2 shows the configuration for setting up the security context, and listings C.3 and C.4 show the definition of WS-Sec protection assertions for incoming and outgoing SOAP messages, respectively.

The policy for the actual message exchange (listing C.1) defines the use of WS-RM (lines 5-7) and WS-Addressing (lines 9-11) as well as the binding for the security context (lines 13-43). The WS-RM assertion activates the use of WS-RM within the security context. Neither [4] nor [44] allow to draw a conclusion about the delivery assurance to be used in the scenario or whether the WS-RM sequence should be bound to a security token using the `wsrmp:SequenceSTR` assertion. Conversely, the use of WS-Addressing is required by both publications. The WS-Addressing assertion enables the use of WS-Addressing message header properties such as `wsa:To`, `wsa:Action`, `wsa:MessageID`, or `wsa:RelatesTo` for implementing functionality as required by the SecRM scenario definition.

The setup of the security context is denoted in the `sp:SymmetricBinding` assertion, since the session key is symmetric. A `sp:SecureConversationToken` is established as protection token. This token is the base for signature and encryption key derivation (`sp:RequireDerivedKeys` assertion). The `sp:BootstrapPolicy` is the policy used to obtain the `sp:SecureConversationToken` from the token issuer (see listing C.2 for a specification of this policy). Within the security context the `sp:Basic128` algorithm is used for encryption, since [4] and [44] propose 128-bit cryptography. Considering the sample message structures in [44], the `sp:Layout` of the SOAP messages is a `sp:Strict` layout (see [145, pages 52/53]) and a `wsu:Timestamp` must be included. Both aspects are not explicitly specified in [4]. A significant difference between the two scenario specifications is the protection order. Whereas [44] states that "signature occurs before encryption" [44, page 35] (first sign the body and then encrypt body and signature), [4] proposes to encrypt the message body first, then to sign the corresponding message parts including the `env:Body`, and finally to encrypt the signature. Since [4] gives a formal cryptographic analysis of this scenario, the `sp:EncryptBeforeSigning` assertion has been chosen. The `sp:EncryptSignature` assertion then requires the encryption of the `wss:Signature`.

Listing C.1: The WS-Policy for the SecRM Scenario

```
<wsp:Policy wsu:Id="SecureRMSessionBinding">
   <wsp:ExactlyOne>
      <wsp:All>

         <wsrmp:RMAssertion>
            <wsp:Policy />
         </wsrmp:RMAssertion>

         <wsam:Addressing>
            <wsp:Policy />
         </wsam:Addressing>

         <sp:SymmetricBinding>
            <wsp:Policy>
               <sp:ProtectionToken>
                  <wsp:Policy>
                     <sp:SecureConversationToken>
                        <wsp:Policy>
                           <sp:RequireDerivedKeys />
                           <sp:BootstrapPolicy>
                              <!--
                              See the XML listing containing the BootstrapPolicy
                              -->
                           </sp:BootstrapPolicy>
                        </wsp:Policy>
                     </sp:SecureConversationToken>
                  </wsp:Policy>
               </sp:ProtectionToken>
               <sp:AlgorithmSuite>
                  <wsp:Policy>
                     <sp:Basic128 />
                  </wsp:Policy>
               </sp:AlgorithmSuite>
               <sp:Layout>
                  <wsp:Policy>
                     <sp:Strict />
                  </wsp:Policy>
               </sp:Layout>
               <sp:IncludeTimestamp />
               <sp:EncryptBeforeSigning />
               <sp:EncryptSignature />
            </wsp:Policy>
         </sp:SymmetricBinding>

      </wsp:All>
   </wsp:ExactlyOne>
</wsp:Policy>
```

Listing C.2 gives the policy for the `sp:BootstrapPolicy` [145, p. 41] of listing C.1. The `sp:BootstrapPolicy` defines the policy for the `sp:SecureConversation-Token` request and the `sp:SecureConversationToken` issuance. [4] and [44] stipulate the use of X509 certificates in the `sp:BootstrapPolicy`, which are certified by a certificate authority. This is important for real-world use cases, however, for the purpose of testing the selected platforms, client and service are in possession of a trusted X509 certificate of each other. This change of the scenario does not have any effect on the policy configuration or the SOAP message traffic between client and service, and is therefore not relevant in this work.

The `sp:BootstrapPolicy` in listing C.2 uses an `sp:AsymmetricBinding` (public-key cryptography) with a `sp:X509Token` for the initiator and the recipient. The initiator should send his public key as `wss:BinarySecurityToken` to the recipient, whereas the key of the recipient must not necessarily be transmitted to the client. The `sp:AlgorithmSuite`, the `sp:Layout`, the `wsu:Timestamp` inclusion, the protection order, and the `sp:EncryptSignature` instruction are used analogously to the `sp:SymmetricBinding` of the main policy.

Listing C.2: The `sp:BootstrapPolicy` of the `sp:SecureConversationToken` in the SecRM Scenario

```
<wsp:Policy>
   <sp:AsymmetricBinding>
      <sp:Policy>
         <sp:InitiatorToken>
            <wsp:Policy>
               <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                   securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                  <wsp:Policy>
                     <sp:WssX509V3Token10 />
                  </wsp:Policy>
               </sp:X509Token>
            </wsp:Policy>
         </sp:InitiatorToken>
         <sp:RecipientToken>
            <wsp:Policy>
               <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                   securitypolicy/200702/IncludeToken/Never">
                  <wsp:Policy>
                     <sp:WssX509V3Token10 />
                  </wsp:Policy>
               </sp:X509Token>
             </wsp:Policy>
          </sp:RecipientToken>
          <sp:AlgorithmSuite>
             <wsp:Policy>
                <sp:Basic128 />
             </wsp:Policy>
          </sp:AlgorithmSuite>
          <sp:Layout>
             <wsp:Policy>
                <sp:Strict />
             </wsp:Policy>
          </sp:Layout>
          <sp:IncludeTimestamp />
          <sp:EncryptBeforeSigning />
          <sp:EncryptSignature />
       </wsp:Policy>
    </sp:AsymmetricBinding>

    <sp:SignedParts>
       <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
       <sp:Header Namespace="http://www.w3.org/2005/08/addressing" />
       <sp:Body />
```

```
42     </sp:SignedParts>
43
44     <sp:EncryptedParts>
45        <sp:Body />
46     </sp:EncryptedParts>
47  </wsp:Policy>
```

In addition to the security binding assertions, the `sp:BootstrapPolicy` specifies the message elements to be protected. The scenario definition stipulates that the WS-Addressing headers, the `wsu:Timestamp`, and the message `env:Body` are to be signed. The assertions in lines 39 and 40 of listing C.2 indicate that all WS-Addressing headers must be signed (for reasons of compatibility the XML namespace of two WS-Addressing versions is specified), the assertion in line 41 defines that the `env:Body` must be signed. The `wsu:Timestamp` of a SOAP message must always be covered by a signature due to the `sp:IncludeTimestamp` definition in [145, page 51]. The encryption of the `env:Body` is declared in lines 44-46 and the encryption of the signature is asserted with the `sp:EncryptSignature` element in the binding (line 34 of listing C.2).

The protection assertions of the messages from the client to the service (see listing C.3) and vice versa (see listing C.4) are similar to the assertions defined in the `sp:BootstrapPolicy`: the WS-Addressing headers and the `env:Body` are intended for integrity protection, and the `env:Body` is also encrypted. In addition, the WS-RM header sections are signed.

Listing C.3: The WS-Security Policy Protection Assertions for the Input Messages

```
<wsp:Policy wsu:Id="SecureRMSessionInput" >
   <sp:SignedParts>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
      <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
      <sp:Header Namespace="http://docs.oasis-open.org/ws-rx/wsrm/200702"/>
      <sp:Body />
   </sp:SignedParts>
   <sp:EncryptedParts>
      <sp:Body />
   </sp:EncryptedParts>
</wsp:Policy>
```

Listing C.4: The WS-Security Policy Protection Assertions for the Output Messages

```
<wsp:Policy wsu:Id="SecureRMSessionOutput" >
   <sp:SignedParts>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
      <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
      <sp:Header Namespace="http://docs.oasis-open.org/ws-rx/wsrm/200702"/>
      <sp:Body />
   </sp:SignedParts>
   <sp:EncryptedParts>
      <sp:Body />
   </sp:EncryptedParts>
</wsp:Policy>
```

C.2. SecRM Scenario Test Results

This section analyzes the SOAP message traffic produced by the selected platforms for the SecRM scenario by checking for the existence of relevant WS-* elements as defined in table C.1 within the exchanged SOAP messages. The approach taken follows the test approach proposed in [197] which is, roughly speaking, invocation of the service by a QoS-aware client, logging of the SOAP message traffic, and analysis of the captured message trace.

The analysis of the test run on the GlassFish-openESB platform shows a close match to the scenario definitions. Yet, the results obtained slightly deviate from the expected results, since, in general, the `wss:Signature` is not encrypted if the `env:Body` is not encrypted on the GlassFish-openESB platform. This concerns message types 3, 4, 7, 8, and 9. Message type 6 contains an encrypted and signed `env:Body`, although the scenario definitions intend an empty and unprotected `env:Body`. Although this is a deviation from the scenario definition, the behavior of the GlassFish-openESB platform is standard compliant since the "*primary signature element is NOT REQUIRED to be encrypted [...] when there is nothing in the message that is covered by this signature that is encrypted.*" [145, lines 1730/1731]. Conversely, the `env:Body` of message type 10 is encrypted, and therefore the `wss:Signature` also is encrypted.

Interestingly, the GlassFish-openESB platform first sends a `wsrm:CloseSequence` message and then a `wsrm:TerminateSequence` message to terminate the sequence. The opposite party answers both messages with a corresponding response message. Although this means that the GlassFish-openESB platform's message sequence differs from the scenario description, this is a minor deviation since both message types are protected in exactly the same way and both have the same purpose. Therefore, the four messages are treated as two two-part message types 7 and 8 in the scenario analysis below.

Table C.2 summarizes the results of the scenario analysis for the GlassFish-openESB platform. It shows the protected message parts and the keys used for protection. The notation conventions correspond to the ones defined for table C.1.

Compared to the GlassFish-openESB platform, the results obtained from the test runs on the IBM WebSphere platform deviate significantly from the scenario definitions. Concerning the message sequence, the fact that neither the WS-RM sequence nor the WS-SecureConversation security context is terminated or canceled is striking. On the IBM WebSphere platform, sequences or sessions have to be closed explicitly by addressing the WS-RM sequence in the source code of a client application. However, this does not fulfill the call for policy-based implementation of WS-* functionality (cf. section 2.1) and is therefore considered to be invalid for this work.

However, not only the message sequence does not comply with the SecRM scenario, but also the structure of several messages. The `wsu:Timestamp` elements have no expiration date, the message `env:Body` always is encrypted, even when the scenario does not intend a confidentiality protection, and the IBM WebSphere platform only uses a `wss:SecurityTokenReference` instead of an embedded `wsc:SecurityContextToken`. Similar to the results for the GlassFish-openESB platform, message type 6 has an integrity and confidentiality protected `env:Body` which is not scenario compliant. Table C.3 lists the protected parts and gives an overview

Msg	**Sign**	**Body**		**TS**	**WS-A**	**WS-R**	**EncKey**
No	**enc**	**enc**	**sig**	**sig**	**sig**	**sig**	**enc**
1	SK_1		PrK_C			$\circ$	PuK_S
2	SK_2		PrK_S			$\circ$	PuK_C
3	$\bullet$	$\bullet$	DK_{C2}			$\circ$	$\circ$
4	$\bullet$	$\bullet$	DK_{S2}			$\circ$	$\circ$
5	DK_{C1}		DK_{C2}				$\circ$
6	DK_{S1}		DK_{S2}				$\circ$
7	$\bullet$	$\bullet$	DK_{C2}				$\circ$
8	$\bullet$	$\bullet$	DK_{S2}				$\circ$
9	$\bullet$	$\bullet$	DK_{C2}			$\circ$	$\circ$
10	DK_{S1}		DK_{S2}			$\circ$	$\circ$

Table C.2.: Message Protection and Keys Used for Protection Realization on the GlassFish-openESB Platform

of the keys used to realize the protection. The notation corresponds to the tables C.1 and C.2.

Msg	**Sign**	**Body**		**TS**	**WS-A**	**WS-R**	**EncKey**
No	**enc**	**enc**	**sig**	**sig**	**sig**	**sig**	**enc**
1	SK_1		PrK_C			$\circ$	PuK_S
2	SK_2		PrK_S			$\circ$	PuK_C
3	DK_{C1}		DK_{C2}			$\circ$	$\circ$
4	DK_{S1}		DK_{S2}			$\circ$	$\circ$
5	DK_{C1}		DK_{C2}				$\circ$
6	DK_{S1}		DK_{S2}				$\circ$

Table C.3.: Message Protection and Keys Used for Protection Realization on the IBM WebSphere Platform

D. SPIN Validation of the BT Execution Model

This appendix contains research results that have been created by Matthias Geiger for his diploma thesis which has been supervised during this work's dissertation project. The contents have been adapted from Matthias' diploma thesis.

The validation of the BT Execution Model is performed using the well-known model checker SPIN[1] that has been invented by Gerard J. Holzmann for the purpose of analyzing interacting software components. From a high-level point of view, validation based on SPIN works as follows. At first, the behavior of the interacting components as well as the message channels used for interaction are captured using SPIN's input language Promela. SPIN is then able to derive the state space of the overall system from the Promela model, that means SPIN derives all possible interactions of the components (as long as the overall system is finite and memory suffices). Relevant properties of the system can be expressed in the temporal logic language *Linear Temporal Logic (LTL)* and SPIN is able to check the corresponding expressions against the state space. The reader is assumed to have a working knowledge of model checking methods and SPIN in particular. For a comprehensive introduction to model checking with SPIN, please see [62]. For another excellent introduction to model checking, please see [17, 25].

The presentation of the SPIN validation of the BT execution model is split up into the following sections. At first, the BT execution model as input for validation is discussed in section D.1. Then, the representation of the BT execution model using Promela is presented in section D.2 and brief description of simulating the model using the XSPIN functionalities is given in section D.3. The formulation of the intended system properties as well as their validation then is presented in section D.4. Section D.5 concludes.

D.1. The BT Execution Model as Validation Input

The object of investigation of the BT execution model are the control process state machines of section 4.3. These specify the admissible message exchanges between the respective control processes, the backend processes and superordinate master processes. Precisely speaking, only the control flow of the control process

[1] `http://spinroot.com/spin/whatispin.html`, last access: 12/20/2011

state machines will be analyzed. The realization of various BT configuration QoS parameters that are proposed to be implemented at the messaging layer (cf. sections 2.3.1 and 4.3) is assumed to be safe. Similarly, the details of implementing the control flow using Web Services and BPEL are not taken into consideration. As a consequence, the validation results of this appendix only apply if the abstract control process models are not modified upon translation into BPEL code.

The sample state machines of figures 4.2 and 4.3 (page 95/96) that represent the most complex Single-Action BT configuration from a control flow point of view are used as input for the formal analysis. This appendix will show that these sample *requester* and *responder* machines eventually terminate in a consistent final state upon execution, i.e., both in the *Success* state or both in the *Failure* state. Strictly speaking, this result cannot be taken to any BT configuration that deviates from the sample state machines' control flow definition. However, it is noteworthy that the validation did not reveal any major flaw (except for naming inconsistencies in the visual models) in the sample machines. So, deriving (less complex) machines for less complex BT configurations should be doable without introducing inconsistencies.

Finally, the sample state machines of section 4.3 do not only define cross-organizational message exchanges, but also message exchanges with *backend* and *master* components. Hence, these must be suitably represented in the validation model. The master components are trivial and the corresponding Promela models therefore will be introduced in the next section. The backend components, however, exchange multiple messages with the control processes and hence are first visualized as state machines in this section.

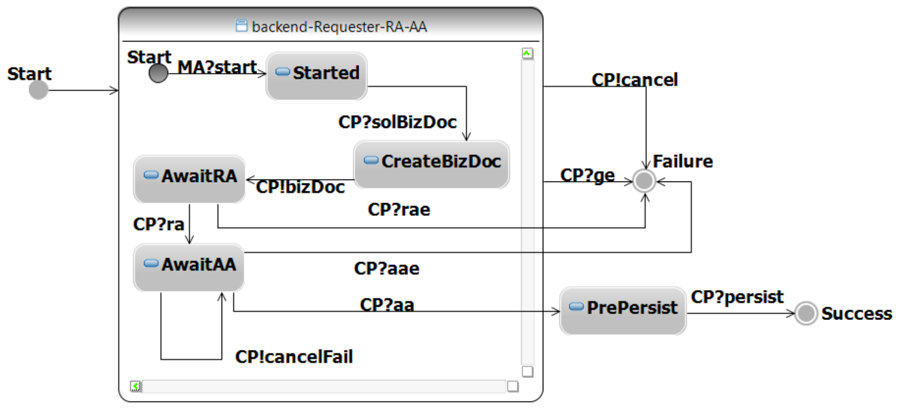


Figure D.1.: *Resquester* Backend Automaton

Figures D.1 and D.2 show the corresponding state machines for the backends of the BT *requester* and *responder* roles, respectively. The visualization concept for the backends corresponds to the visualization concept of the control process state machines, that means the labels for states and transitions are defined by analogy. Moreover, the concept for unfolding hierarchical states is the same. In the start state of the backend machines, receiving a *start* message from the master components is admissible. After that, the exchange of business documents and business signals as

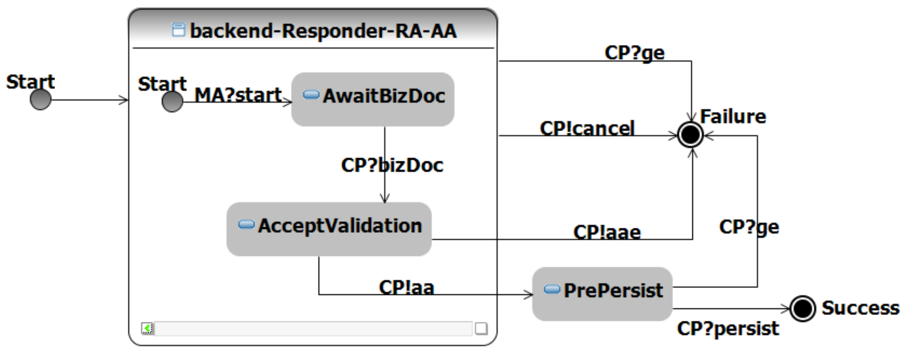


Figure D.2.: *Responder* Backend Automaton

implied by the corresponding control process machines takes place. In that regard, it is noteworthy that there are no events in the backend machines that represent failing message exchanges. This is because communication between control processes and backends is assumed to be safe (cf. section 4.2). The details of transforming these state machines into a Promela representation are given in the next section.

D.2. Promela Representation of the BT Execution Model

This section discusses the assumptions, design choices and language constructs used for transforming the state machines of the BT execution model into valid Promela processes. First, the overall structure of the process system, global variable definitions as well as the message channels used are described. Afterwards, the individual processes are described in detail.

D.2.1. Overall Process Structure and Global Resources

For each component of the BT execution model a separate Promela `proctype` (process) is defined that represents the behavior of the respective component. This leads to the following list of `proctypes`:

- The *Requester* control process (REQ)
- The *Responder* control process (RESP)
- The *Requester* backend process (BEreq)
- The *Responder* backend process (BEresp)
- The *Requester* master process (MAreq)
- The *Responder* master process (MAresp)

- The service for creating RA signal messages (ReceiptAcknowledgementCreator (RAC))

The goal of the validation is to prove that all backend and control processes eventually reach a common final state, that means all processes are supposed to terminate in the *Failure* state or, in case the BT execution is successful, in the *Success* state. Therefore, boolean variables are added to each of the corresponding `proctypes` in order to represent success or failure (`*processName*EndStateFail` and `*processName*EndStateSuccess`, respectively).

In addition, Promela `channels` are defined for representing the messaging channels between the `proctypes`. In that regard, it is vital to note that synchronous communication is assumed between the interacting components. As a consequence, the Promela `channels` are defined to have a buffer length of 0. The identifiers for the `channels` are derived according to the pattern `chan*sender*2*receiver*`. Moreover, it is noteworthy that each `channel` is unidirectional so that the interaction between two `proctypes` necessitates the definition of two `channels` in opposite directions.

In the Promela model, messages are represented as symbolic names (declared using the `mtype` keyword). This simplification is unproblematic for the BT execution model because it does not rely on the message contents (as opposed to the control flow of BCs). Therefore, the message names can simply be taken over from the state machine definitions of the BT execution model. Listing D.1 shows the definition of all message and channel definitions.

Listing D.1: Promela Definition of Messages and Message Channels

```
/* Global constants */
/* Constant representing the retryCount parameter of the BT
   execution model */
#define MAX_RETRIES 3

/* Global variables */
/* Boolean variables for representing the Failure end state
   of proctypes */
bool reqEndStateFail, respEndStateFail, reqBEEndStateFail,
   respBEEndStateFail;

/* Boolean variables for representing the Success end state
   of proctypes */
bool reqEndStateSuccess, respEndStateSuccess,
   reqBEEndStateSuccess, respBEEndStateSuccess;

/* Message type declaration */
mtype = {
 start,
 generalException,
 cancel,
 solicitBizDoc,
 sendBizDoc,
 receiptAckException, receiptAck,
 acceptAckException, acceptAck,
 persistStateChanges
};

/* Message channel declaration */
chan req2resp = [0] of { mtype };
chan resp2req = [0] of { mtype };

chan req2be = [0] of { mtype };
chan be2req = [0] of { mtype };

chan resp2be = [0] of { mtype };
chan be2resp = [0] of { mtype };

chan resp2rac = [0] of { mtype };
chan rac2resp = [0] of { mtype };

chan ma2req = [0] of { mtype };
chan ma2resp = [0] of { mtype };

chan ma2beReq = [0] of { mtype };
chan ma2beResp = [0] of { mtype };
```

D.2.2. Promela Representation of the Requester Control Process

Before the Promela representation of the *requester* control process is given, the approach for transforming the state machines of the BT execution model is described.

Promela does not offer an explicit concept for modeling states. As a consequence the automaton states such as *Started* or *AwaitBizDoc* are only implicitly existent in the below `proctypes`. For better readability, the names of the automaton states will added in comments. In contrast to the states, the transitions of the state machines can directly be transformed into Promela code. For example, the transition labeled `BE?cancel` of the requester's state machine can be encoded as `be2req?cancel` in Promela. As the Promela processes use symbolic names for representing message types, the message names of the state machines can be taken over unmodified. In addition, the syntax and semantics of representing incoming (`?`) and outgoing (`!`) messages is the same for Promela and the used state machine formalization so that the symbols '`?`' and '`!`' also can be taken over unmodified. In contrast to this, the names of the messaging partners as used in the state machine model cannot be taken over unmodified because Promela enforces using `channels` for communication. Hence, the name of the communication partner must be replaced with the name of the communication `channel` that is defined for the two communication partners under consideration. For example, if the *requester* process sends a message to the *backend* process then the channel *req2be* is to be used. If the *requester* receives a message from the *backend* then the channel *be2req* is to be used.

The state machines of the BT execution model require the definition of branching behavior. For example, in the *Started* state of the *requester* control process either sending a *solicitBizDoc* message to the *responder*, receiving a *generalException* from the *responder*, receiving a *cancel* message from the *backend* or the *TimeToPerform* timeout are admissible. The Promela `if` construct can be used to capture such branching behavior. However, it is vital to note that the semantics of a Promela `if` significantly deviates from the semantics of an `if` construct in procedural programming languages. In Promela, the guards of the branches of the `if` construct (defined as `::guard -> ...`) are evaluated first. Then, one branch is selected on a non-deterministic basis out of those branches with a guard that evaluates to *true* (cf. [62, pages 56-58 and 424-425]). For demonstrating the concept, listing D.2 shows the representation of the *Started* state of the requester control process.

Listing D.2: Exemplary Usage of a Promela `if` Construct

```
...
/* State: Started */
if
 ::resp2req?generalException ->
  propagateErrorToBackend(req2be,reqEndStateFail);
 ::be2req?cancel ->
  propagateError(req2resp,reqEndStateFail);
 ::true -> /* Enable Timeout TimeToPerform*/
  propagateErrorToBackend(req2be,reqEndStateFail);
  propagateError(req2resp,reqEndStateFail);
 ::req2be!solicitBizDoc ->
  /* State: AwaitBizDoc */
  if ... fi;
fi;
...
```

The attentive reader may have noticed that listing D.2 does not define any kind of counter or clock for controlling the *TimeToPerform* timer. Indeed, Promela abstracts from *real time* or *clocks*. The concept of a timeout is instead modeled by means of a `true`-guarded branch. This basically means that the timeout can be fired irrespective of how fast the alternative activities are performed. Note that this reflects the nature of distributed systems very well because well-defined distributed systems should not rely on timing. Moreover, the probability of a timeout is not of any interest for the analysis of the BT execution model because it is supposed to *always* produce correct results and not only in the *majority of runs*.

Timeouts may occur in almost any state of the BT execution model's state machines. Similarly, *cancel* messages of the backend or *exception* messages of the partner control process also are accepted in almost any state. As the reactions to such events is almost always the same, Promela `inline` definitions are used to maintain well-structured Promela code. A Promela `inline` definition is not a function or procedure as available in procedural programming languages. Instead, an `inline` definition declares the name and the parameters of a textual template that can be included in different Promela code by referencing the name and providing values for the parameters (cf. [62, pages 64-66 and 428-429]). Listing D.3 shows the usage of `inline` definitions for the purpose of reusing the behavioral specification of processing timeouts or exception messages. The following list enumerates the different cases that can be covered based on these templates:

1. **The backend of a particular control process terminates the BT using a `cancel` message.** In this case, the respective control process is in charge of informing the partner control process about the termination. It is noteworthy that sending the corresponding exception message to the partner control process may fail. For example, the partner control process may detect an error at the same time and try to terminate the BT execution as well. In order to avoid blocking control processes in such a case, the Promela '`timeout`' variable is used. The Promela `timeout` is a predefined global boolean variable that becomes true whenever no further action is possible in the system under consideration. Hence, using the `timeout` variable as shown in listing D.3 is a means to model the *geFail* events of the control process state machines. In any case, that means whether the `generalException` message has been delivered to the partner control process or not, the respective *Failure* state variable of the control process is set to *true*.
   The representation of this behavior is shown in the `inline` definition `propagateError(msgchan, stateVar)` of the listing D.3. The parameters to be provided are the message `channel` to be used and the state variable to be written.

2. **A generalException message is received from the partner control process.** In this case, the respective control process is in charge of informing the corresponding backend about the failure. The representation of this case is similar to the first case, but the predefined Promela `timeout` variable is not needed. This is due to the fact that the backend cannot terminate without informing its corresponding control process.

3. **The control process itself detects the need for terminating the BT execution.** Valid reasons for which a control process may decide to terminate a BT execution are timeouts or repeatedly failing business document/business signal transmissions. In this case, the corresponding backend as well as the partner control process have to be informed about the BT execution termination. For representing this case in Promela, the `inline` definitions for the first two cases can be used.

Listing D.3: The *inline* Definitions Used

```
inline propagateError(msgchan, stateVar)
{
 if
  ::msgchan!generalException;
  ::timeout -> skip; /* generalException cannot be delivered
      */
 fi;
 stateVar = true;
}

inline propagateErrorToBackend(msgchan, stateVar)
{
 msgchan!generalException;
 stateVar = true;
}
```

Finally, listing D.4 shows the Promela definition of the requester control process that has been derived according to the above procedures.

Listing D.4: Promela Definition of the *Requester* Control Process

```
/* Requester Control Process */
proctype requester()
{
 int retryCount = 0;
 ma2req?start;
 /* State: Started */
 if
  ::resp2req?generalException ->
   propagateErrorToBackend(req2be,reqEndStateFail);
  ::be2req?cancel ->
   propagateError(req2resp,reqEndStateFail);
  ::true -> /* Enable Timeout TimeToPerform*/
   propagateErrorToBackend(req2be,reqEndStateFail);
   propagateError(req2resp,reqEndStateFail);
  ::req2be!solicitBizDoc ->
   /* State: AwaitBizDoc */
   if
    ::resp2req?generalException ->
     propagateErrorToBackend(req2be,reqEndStateFail);
    ::be2req?cancel ->
     propagateError(req2resp,reqEndStateFail);
    ::true -> /* Enable Timeout TimeToPerform*/
     propagateErrorToBackend(req2be,reqEndStateFail);
     propagateError(req2resp,reqEndStateFail);
    :: be2req?sendBizDoc ->
    /* State: DeliverBizDoc */
```

```
     do
      ::be2req?cancel ->
       propagateError(req2resp,reqEndStateFail);
       break;
      ::resp2req?generalException ->
       propagateErrorToBackend(req2be,reqEndStateFail);
       break;
      ::true -> /* Enable Timeout TimeToPerform*/
       propagateErrorToBackend(req2be,reqEndStateFail);
       propagateError(req2resp,reqEndStateFail);
       break;
      ::(retryCount <= MAX_RETRIES) -> retryCount = retryCount + 1; /* Simulate undeliverable message */
      ::(retryCount>MAX_RETRIES) -> /* Maximum number of retries exceeded */
       propagateErrorToBackend(req2be,reqEndStateFail);
       propagateError(req2resp,reqEndStateFail);
       break;
      ::req2resp!sendBizDoc ->
       /* State: AwaitReceiptAck */
       if
        ::be2req?cancel ->
         propagateError(req2resp,reqEndStateFail);
        ::resp2req?generalException ->
         propagateErrorToBackend(req2be,reqEndStateFail);
         break;
        ::true -> /* Enable Timeouts TimeToPerform, TimeToReceiptAck, TimeToAcceptAck*/
         propagateErrorToBackend(req2be,reqEndStateFail);
         propagateError(req2resp,reqEndStateFail);
        ::resp2req?receiptAckException ->
         /* State: DeliverReceiptAckException */
         if
          ::be2req?cancel ->
           propagateError(req2resp,reqEndStateFail);
          ::true -> /* Enable Timeout TimeToPerform*/
           propagateErrorToBackend(req2be,reqEndStateFail);
           propagateError(req2resp,reqEndStateFail);
          ::req2be!receiptAckException;
         fi;
         reqEndStateFail = true;
        ::resp2req?receiptAck ->
         /* State: DeliverReceiptAck */
         if
          ::be2req?cancel ->
```

```
           propagateError(req2resp,reqEndStateFail);
          ::resp2req?generalException ->
           propagateErrorToBackend(req2be,reqEndStateFail);
          ::true -> /* Enable Timeouts TimeToPerform, TimeToAcceptAck */
           propagateErrorToBackend(req2be,reqEndStateFail);
           propagateError(req2resp,reqEndStateFail);
          ::req2be!receiptAck ->
           /* State: AwaitAcceptAck */
           if
            ::be2req?cancel ->
             propagateError(req2resp,reqEndStateFail);
            ::resp2req?generalException ->
             propagateErrorToBackend(req2be,reqEndStateFail);
            ::true -> /* Enable Timeouts TimeToPerform, TimeToAcceptAck */
             propagateErrorToBackend(req2be,reqEndStateFail);
             propagateError(req2resp,reqEndStateFail);
            ::resp2req?acceptAckException ->
             /* State: DeliverAcceptAckException */
             if
              ::be2req?cancel ->
               propagateError(req2resp,reqEndStateFail);
              ::true -> /* Enable Timeout TimeToPerform*/
               propagateErrorToBackend(req2be,reqEndStateFail);
               propagateError(req2resp,reqEndStateFail);
              ::req2be!acceptAckException;
               reqEndStateFail = true;
             fi;
            ::resp2req?acceptAck ->
             /* State: DeliverAcceptAck */
             req2be!acceptAck;
             /* State: Propagate */
             req2be!persistStateChanges;
             reqEndStateSuccess = true;
           fi;
         fi;
       fi;
       break;
     od;
   fi;
 fi;
}
```

D.2.3. Promela Representation of the Responder Control Process

The Promela representation of the *responder* control process can basically be derived by analogy with the *requester's* control process representation. A major difference, though, is that timeouts are mainly controlled by the *requester* control process so that `true`-guarded branches of `if` constructs are not needed. The only exception is the *responder's AwaitBizDoc* state in which a timeout is used to terminate the process in case the *requester* never sends the business document.

Another important specialty of the *responder* process is that sending the *ReceiptAcknowledgement* message to the *requester* and receiving the *AcceptanceAcknowledgement* from the *backend* can occur concurrently. After the RAC service has successfully validated the business document, it will be delivered to the backend. The backend, in turn, checks the business document for processability and correspondingly answers with an *AcceptanceAcknowledgement* or an *AcceptanceAcknowledgementException*. At the same time, the *responder* control process tries to deliver the *ReceiptAcknowledgement* to the *requester* control process. These activities are defined to be executed concurrently so that the oder of sending/receiving the corresponding messages in the *responder* control process is not determined. Either the *ReceiptAcknowledgement* is sent first and then the *AcceptanceAcknowledgement* is received or the other way round. In addition, the transmission of the *ReceiptAcknowledgement* may have to be repeated in case of delivery errors and exception/cancel messages may be received from the *requester* or from the *backend* process, respectively. The responder automaton states *DeliverRA*, *AwaitAcceptance*, *Deliver RA-AA* and *GotAcceptAckException* correspond to this concurrent behavior. Their Promela representation is shown in lines 49-79 of listing D.5. It is noteworthy that these states are not represented by analogy with the other states. Instead, they are enclosed within a Promela `do` loop that is performed until the *ReceiptAcknowledgement* has successfully been transmitted (or the *retryCount* is exceeded) and the *AcceptanceAcknowledgement* has been received. In order to prevent multiple successful transmissions of the *ReceiptAcknowledgement* or multiple receipts of the *AcceptanceAcknowledgement*, boolean flags are used to represent successful transmission or receipt (see lines 49/50 of listing D.5). Upon successful transmission/receipt the flags are set to *true* which disables the corresponding branches of the Promela `do` loop. Once both flags have been set to `true` the *DeliverAcceptanceAck* state of the *responder* state machine is reached the representation of which starts in line 82. In this state, the *AcceptanceAcknowledgement* is tried to be transmitted to the *requester* process. For this purpose, the *retryCount* is reset to 0 beforehand (see line 81).

Apart from the two peculiarities just described the Promela model of the responder process is similar to the requester process. Listing D.5 shows the Promela code for representing the `responder` control process.

Listing D.5: Promela Definition of the *Responder* Control Process

```
/* Responder Control Process */
proctype responder()
{
 int retryCount = 0;
 ma2resp?start;
 /* State: AwaitBizDoc */
 if
  ::req2resp?generalException ->
   propagateErrorToBackend(resp2be,respEndStateFail);
  ::be2resp?cancel ->
   propagateError(resp2req,respEndStateFail);
  ::true -> /* Enable Timeout TimeToPerform*/
   propagateErrorToBackend(resp2be,respEndStateFail);
   propagateError(resp2req,respEndStateFail);
  ::req2resp?sendBizDoc ->
  /* State: GotBizDoc */
   if
    ::req2resp?generalException ->
     propagateErrorToBackend(resp2be,respEndStateFail);
    ::be2resp?cancel ->
     propagateError(resp2req,respEndStateFail);
    ::resp2rac!sendBizDoc; ->
     /* State: AwaitValidation */
     if
      ::req2resp?generalException ->
       propagateErrorToBackend(resp2be,respEndStateFail);
      ::be2resp?cancel ->
       propagateError(resp2req,respEndStateFail);
      ::rac2resp?receiptAckException ->
       /* State: GotReceiptAckException */
       if
        ::resp2req!receiptAckException;
        ::timeout -> skip; /* Simulate delivery error */
       fi;
       if
        ::resp2be!receiptAckException;
        ::timeout ->skip; /* Simulate delivery error */
       fi;
       respEndStateFail = true;
      ::rac2resp?receiptAck ->
       /* State: GotReceiptAck */
       if
        ::req2resp?generalException ->
         propagateErrorToBackend(resp2be,respEndStateFail);
        ::be2resp?cancel ->
```

```
         propagateError(resp2req,respEndStateFail);
        ::resp2be!sendBizDoc ->
         /* State: DeliverReceiptAck */
         bool sendRAsuccessful = false;
         bool receivedAcceptAck = false;
         do
          ::req2resp?generalException ->
           propagateErrorToBackend(resp2be,respEndStateFail);
           break;
          ::!sendRAsuccessful ->
           if
            ::(retryCount <= MAX_RETRIES) -> retryCount = retryCount + 1; /* Simulate delivery error */
            ::(retryCount>MAX_RETRIES) ->
             propagateErrorToBackend(resp2be,respEndStateFail);
             propagateError(resp2req,respEndStateFail);
             break;
            ::resp2req!receiptAck -> sendRAsuccessful = true;
           fi;
          ::!receivedAcceptAck ->
           if
            /* Backend: Canceling is allowed for unless the AA signal has been sent */
            ::be2resp?cancel ->
             propagateError(resp2req,respEndStateFail);
             break;
            ::be2resp?acceptAckException ->
             if
              ::resp2req!acceptAckException;
              ::timeout -> skip;
             fi;
             respEndStateFail = true;
             break;
            ::be2resp?acceptAck -> receivedAcceptAck = true;
           fi;
          ::sendRAsuccessful && receivedAcceptAck ->
           /* State: DeliverAcceptanceAck */
           retryCount = 0; /* Reset retryCount */
           do
            ::req2resp?generalException ->
             propagateErrorToBackend(resp2be,respEndStateFail);
             break;
            ::(retryCount <= MAX_RETRIES) -> retryCount =
               retryCount + 1; /* Simulate delivery error */
            ::(retryCount>MAX_RETRIES) ->
             propagateErrorToBackend(resp2be,
                respEndStateFail);
             propagateError(resp2req,respEndStateFail);
             break;
            ::resp2req!acceptAck ->
             /* State: Propagate */
             resp2be!persistStateChanges;
             respEndStateSuccess = true;
             break;
           od;
           break;
         od;
       fi;
     fi;
   fi;
 fi;
}
```

D.2.4. Promela Representation of the Requester's Backend Process

The derivation of the Promela representation of the *requester's backend* process by and large corresponds to the above description. Alone the *AwaitAcceptAck* state deserves more detailed discussion.

Once the *requester's backend* has entered this state, canceling the BT may not be admissible any more depending on whether the *requester* control process has received the *AcceptanceAcknowledgement* signal yet or not. If the *requester* control process has received the *AcceptanceAcknowledgement*, canceling is not admissible any more. In case the backend tries to transmit a *cancel* message nonetheless, the corresponding *cancelFail* event is fired in the backend's state machine. Representing this behavior in Promela is not a big issue because synchronous communication is assumed, that means the channel buffer has size 0. This, in turn, implies that the success of the *cancel* message transmission depends on whether the *requester* control process accepts it or not. In case canceling fails, the *requester's backend* process remains in the same state. This case is represented using a `do` loop and a `::true` guard (see line 27 in listing D.6). The problem with this representation is that the `do` loop may be repeated infinitely often which is inconvenient for the validation. In order to enable validation, the two progress labels `inLoop` and `passedLoop` are defined that will be discussed in more detail during the presentation of the validation in section D.4.

Listing D.6: Promela Definition of the *Requester* Backend Process

```
/* Requester backend process */
proctype BEreq(){
 ma2beReq?start;
 /* State: Started */
 if
  ::be2req!cancel -> reqBEEndStateFail = true;
  ::req2be?generalException -> reqBEEndStateFail = true;
  ::req2be?solicitBizDoc ->
  /* State: createBizDoc */
   if
    ::be2req!cancel -> reqBEEndStateFail = true;
    ::req2be?generalException -> reqBEEndStateFail = true;
    ::be2req!sendBizDoc ->
    /* State: AwaitReceiptAck */
     if
      ::be2req!cancel -> reqBEEndStateFail = true;
      ::req2be?generalException -> reqBEEndStateFail = true;
      ::req2be?receiptAckException -> reqBEEndStateFail =
         true;
      ::req2be?receiptAck ->
      /* State: AwaitAcceptAck */
       do
        ::be2req!cancel ->
         reqBEEndStateFail = true;
         break;
        ::true -> /* 'cancelFail': canceling failed - stay
           in the same state */
inLoop:       skip;
        ::req2be?generalException ->
         reqBEEndStateFail = true;
         break;
        ::req2be?acceptAckException ->
         reqBEEndStateFail = true;
         break;
        ::req2be?acceptAck ->
        /* State: PrePersist */
         req2be?persistStateChanges;
         reqBEEndStateSuccess = true;
         break;
       od;
passedLoop:   skip;
     fi;
   fi;
 fi;}
```

D.2.5. Promela Representation of the Responder's Backend Process

The Promela representation of the responder's backend process can be derived by analogy with the above Promela processes. Listing D.7 therefore shows its definition without further discussion.

Listing D.7: Promela Definition of the *Responder* Backend Process

```
/* Responder Backend Process */
proctype BEresp()
{
 ma2beResp?start;
 /* State: AwaitBizDoc */
 if
  ::resp2be?generalException -> respBEEndStateFail = true;
  ::be2resp!cancel -> respBEEndStateFail = true;
  ::resp2be?receiptAckException -> respBEEndStateFail = true
     ;
  ::resp2be?sendBizDoc ->
  /* State: AcceptValidation */
   if
    ::resp2be?generalException -> respBEEndStateFail = true;
    ::be2resp!cancel -> respBEEndStateFail = true;
    ::be2resp!acceptAckException -> respBEEndStateFail =
       true;
    ::be2resp!acceptAck ->
    /* State: PrePersist */
     if
      ::resp2be?generalException -> respBEEndStateFail =
         true;
      ::resp2be?persistStateChanges -> respBEEndStateSuccess
          = true;
     fi
   fi
 fi
}
```

D.2.6. Promela Representation of the Master Processes

The only purpose of the master processes as used in the BT execution model is starting the backend processes and the control processes. The Promela representation of the master processes is correspondingly simple. At first, a *start* message is sent to the respective backend process and then a *start* message is sent to the corresponding control process.

Listing D.8: Promela Definition of the Master Processes

```
proctype MAreq()
{
 ma2beReq!start;
 ma2req!start;
}

proctype MAresp()
{
 ma2beResp!start;
 ma2resp!start;
}
```

D.2.7. Promela Representation of the ReceiptAcknowledgementCreation Service

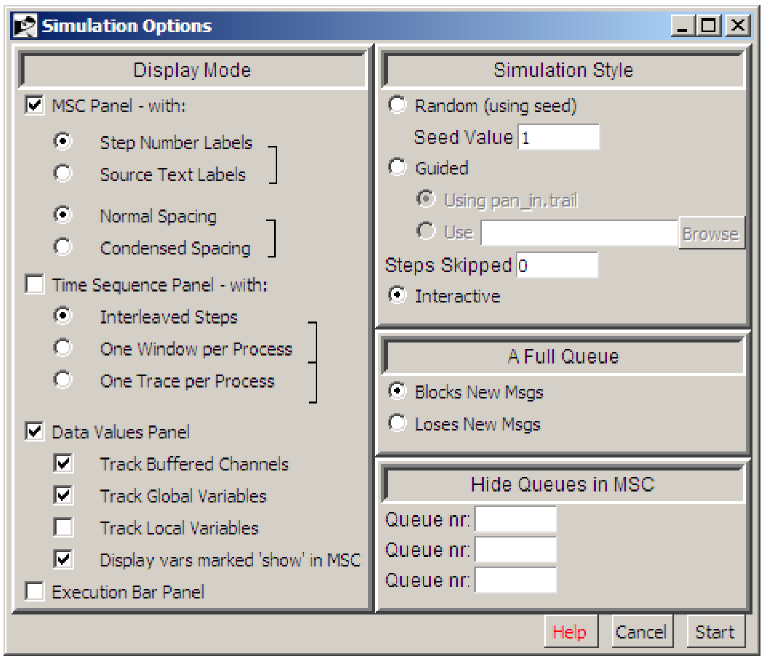
The behavior of the *ReceiptAcknowledgementCreation service* (RAC) is rather simple. Upon incoming request, the RAC validates the corresponding business document and replies with a *ReceiptAcknowledgement* or a *ReceiptAcknowledgementException* depending on the validation result. This choice can be represented using a Promela `if` construct that non-deterministically selects between the two different results as shown in listing D.9. In addition to the two reply options of the RAC a third branch leveraging Promela's `timeout` variable is added to the `if` construct in order to avoid blocking the process (cf. above). This is necessary because the control processes may terminate before the RAC has provided a result. Moreover, the BT execution may be terminated before the RAC even has been called. From an application point of view, this is not critical because the RAC does not store any state and therefore does not have to be aligned with the result of the BT execution. From the point of view of Promela validation, this scenario is critical because the RAC would represent a blocking process. Hence, a so-called *end label* (`end_cancelBeforeCallingRAC`; cf. [62, pages 76-78 and 413-414]) is added to the Promela definition of the RAC service in order to declare that staying in the initial state is acceptable for the RAC process.

Listing D.9: Promela Definition of the ReceiptAcknowledgementCreation Service (RAC)

```
/* ReceiptAck-Creation-Service process */
proctype recAckService()
{
end_cancelBeforeCallingRAC:
 resp2rac?sendBizDoc;
 if
  :: rac2resp!receiptAck;
  :: rac2resp!receiptAckException;
  :: timeout -> skip;
 fi
}
```

D.3. BT Execution Model Simulation Using XSPIN

XSPIN is the graphical user interface of the SPIN system. An important XSPIN feature for exploring the behavior of the specified system is simulation. XSPIN offers to either randomly generate sample runs (also denoted as *Trail*) of the system or to let the user choose between alternative paths in an interactive manner. In addition, XSPIN offers different options for visualizing system runs, among others *Timeline* and *Message Sequence Charts*. Moreover, the values of the local and global variables can be displayed. Figure D.3 shows the selected options that have been used for the following example runs.



Simulation Options

Display Mode
- [✓ MSC Panel - with:]
  - [✓ Step Number Labels]
  - Source Text Labels
  - [✓ Normal Spacing]
  - Condensed Spacing
- Time Sequence Panel - with:
  - [✓ Interleaved Steps]
  - One Window per Process
  - One Trace per Process
- [✓ Data Values Panel]
  - [✓ Track Buffered Channels]
  - [✓ Track Global Variables]
  - Track Local Variables
  - [✓ Display vars marked 'show' in MSC]
- Execution Bar Panel

Simulation Style
- Random (using seed)
  - Seed Value 1
- Guided
  - [✓ Using pan_in.trail]
  - Use
- Steps Skipped 0
- [✓ Interactive]

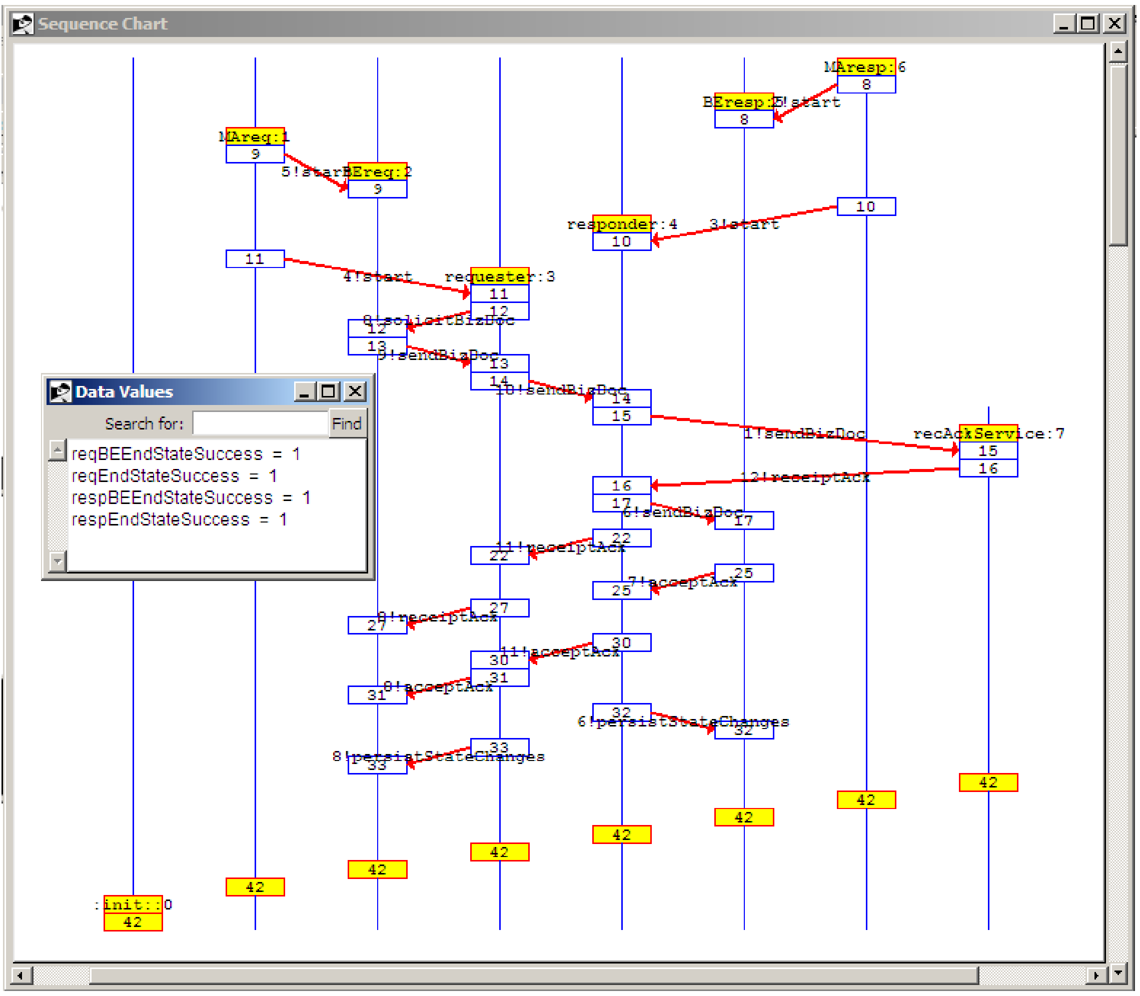
A Full Queue
- [✓ Blocks New Msgs]
- Loses New Msgs

Hide Queues in MSC
- Queue nr:
- Queue nr:
- Queue nr:

Figure D.3.: Simulation Configuration for the Process System (Screenshot XSPIN)

Figure D.4 shows an exemplary successful run, that means an interaction that does not terminate prematurely because of *Receipt*- or *AcceptanceAcknowledgementExceptions*. After the BT execution protocol simulation terminates successfully, all local process state variables signifying *Success* are set to true (cf. figure D.4).

Figure D.5 shows a system run that contains the transmission of an *AcceptanceAcknowledgementException*. After the business document has been received by the *responder* control process, it is passed on to the RAC for legibility validation. Then, the RAC confirms legibility so that the business document is passed on to the back-end system that reacts with an *AcceptanceAcknowledgementException* which is then



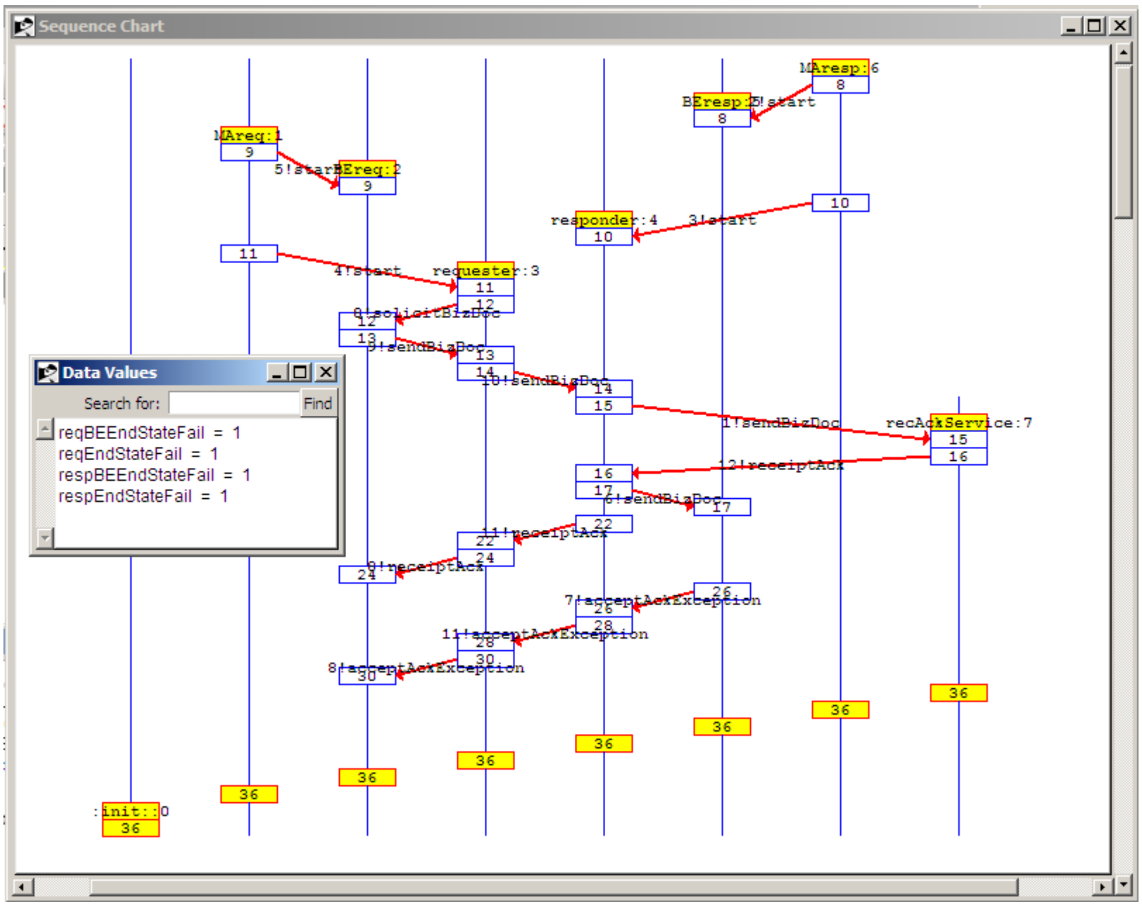
```
reqBEEndStateSuccess = 1
reqEndStateSuccess = 1
respBEEndStateSuccess = 1
respEndStateSuccess = 1
```

Figure D.4.: `Message Sequence Chart` of a Successful Run (Screenshot XSPIN)

passed on to the *requester* control process and the *requester backend* subsequently. At the end of the run, all local process state variables signifying *Failure* are set to true as intended.

D.4. Validation of the BT Execution Model Using SPIN

In its default configuration, SPIN verifies *absence of invalid end states* as a basic soundness property of systems. Therefore, if any `proctype` reaches an end state and if the overall system reaches an end state, that means no further action is possible, then the end state of the respective `proctype` must be valid. If not defined otherwise, the only valid end state of a `proctype` is the end of its code which is demarcated by the closing curly brace of its definition body (cf. [62, page 77]). However, this default notion of valid end state is not adequate for processes that are to be repeatedly used,



```
reqBEEndStateFail = 1
reqEndStateFail = 1
respBEEndStateFail = 1
respEndStateFail = 1
```

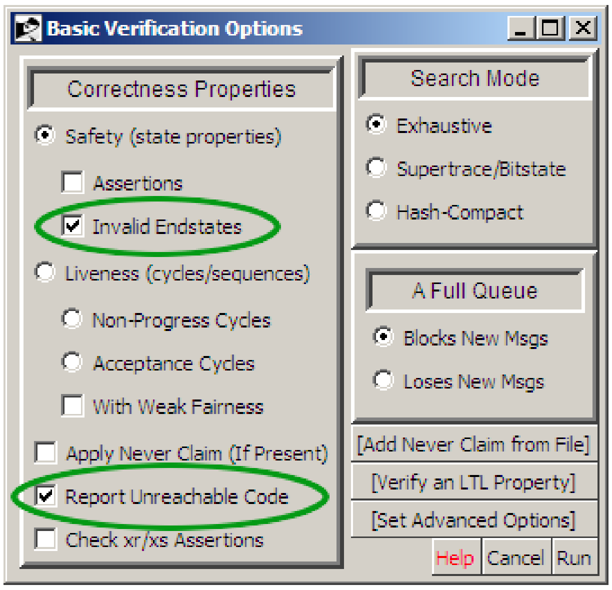
Figure D.5.: `Message Sequence Chart` of an Erroneous Run (Screenshot XSPIN)

for example semaphores. In addition, other processes may have to be performed on an optional basis only. Then, the beginning of the corresponding `proctype` definition would be a valid end state as well which would correspond to not performing the respective process at all. The RAC process is an example for optional behavior because the interaction between control processes may terminate before the RAC has been called (which is a valid system run as defined in the BT execution model). In order to define additional valid end states for processes, Promela offers so-called *end labels* (cf. above). Hence, a corresponding end label is also defined for the RAC process definition (cf. listing D.9, row 4).

In order to verify *absence of invalid end states*, the configuration as shown in figure D.6 has been used. The corresponding verification result is shown in figure D.7.

As SPIN does not report on *invalid end states*, there is no final system configuration that comprises a `proctype` in an invalid end state. This also implies absence of *deadlocks* as a deadlock would imply `proctypes` in end states (that means no further action is possible) that are not valid.

In addition, SPIN allows for analyzing unreachable code. In order to activate this type of analysis, the option *“Report Unreachable Code”* must be activated in the



Basic Verification Options

Correctness Properties

[✓ Safety (state properties)]

Assertions

[✓ Invalid Endstates]

Liveness (cycles/sequences)

Non-Progress Cycles

Acceptance Cycles

With Weak Fairness

Apply Never Claim (If Present)

[✓ Report Unreachable Code]

Check xr/xs Assertions

Search Mode

[✓ Exhaustive]

Supertrace/Bitstate

Hash-Compact

A Full Queue

[✓ Blocks New Msgs]

Loses New Msgs

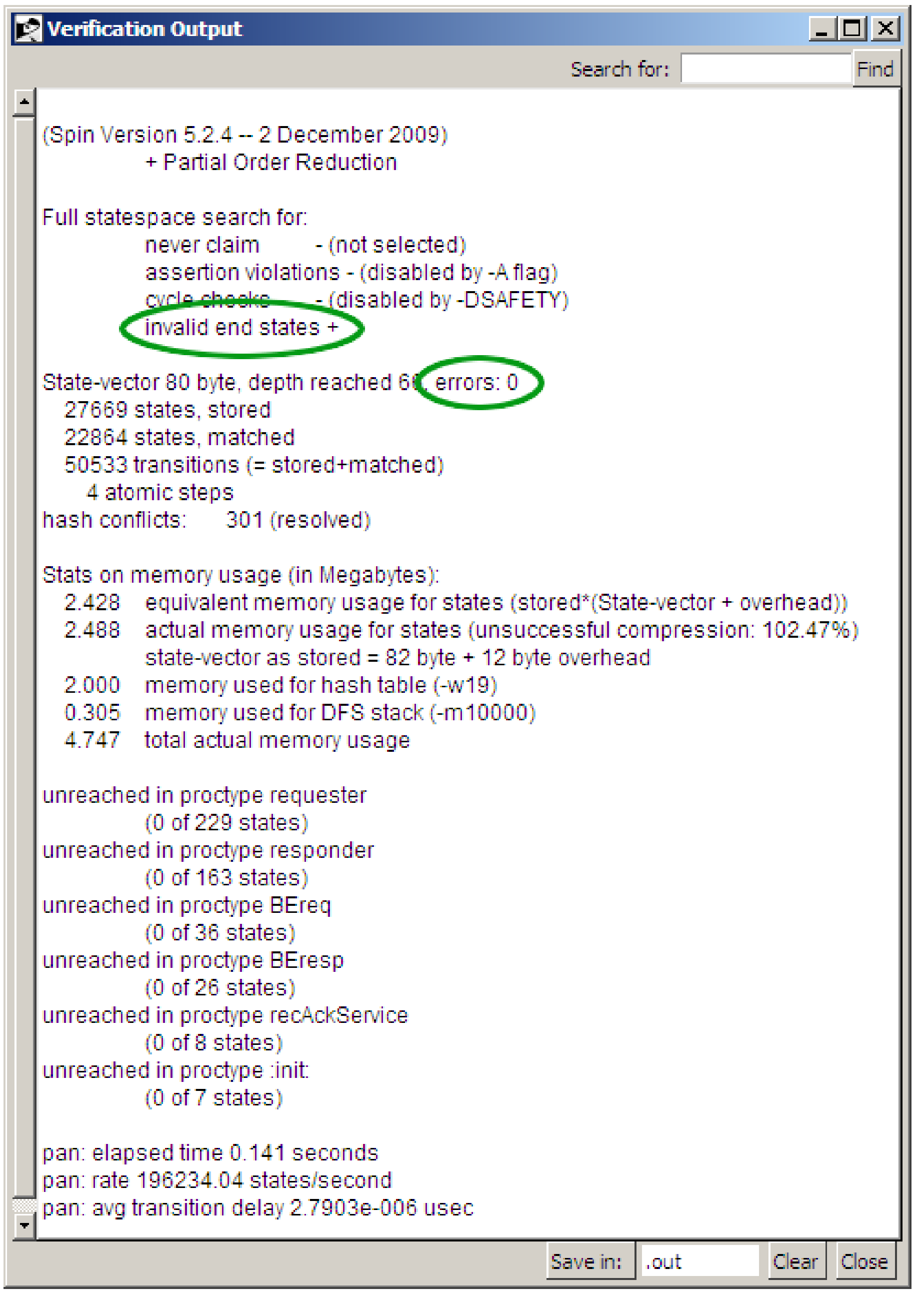
Figure D.6.: SPIN Configuration for Invalid End States Analysis (Screenshot XSPIN)

validation configuration (cf. figure D.6). As SPIN allows for analyzing the whole state space of the system under validation, the individual actions of the `proctype` declarations can be checked for containment in the state space. SPIN has been configured to report *unreachable code* when validating the BT execution model. The result is that all states of the execution model's state machines are reachable (cf. figure D.7).

Beyond *safety properties* such as *absence of invalid end states*, SPIN is also able to verfiy so-called *liveness properties* (cf. [17, 25]). A famous example of a liveness property is *absence of non-progress cycles*. A *non-progress cycle* is a cycle that can be potentially executed infinitely often. The *requester backend's* state machine contains such a non-progress cycle because it may try to cancel the current BT infinitely often in state *AwaitAcceptanceAck* (represented as `do` loop in lines 22-39 of listing D.6).

In that regard, the notions of *weak fairness* and *strong fairness* are of relevance. Assume that a particular process is permanently able to perform a particular action. Then, *weak fairness* means that the process *eventually* will perform that particular action. The difference to *strong fairness* is that enabling the action infinitely often (that means not necessarily permanently) implies that the action *eventually* will be performed.

Intuitively, a *fair* process execution would imply that the *requester backend's* state machine *eventually* will not perform the *cancelFail* transition, but will receive an exception message or an *AcceptanceAcknowledgement*. To be more precise, *strong fairness* is needed which is not supported by SPIN directly. *Strong fairness* is needed because no action of the *requester backend's* non-progress cycle is permanently enabled. For example, receiving an *AcceptanceAcknowledgement* (`req2be?acceptAck`) is not enabled as long as the statements `::true -> /* 'cancelFail': cancel-`



```
(Spin Version 5.2.4 -- 2 December 2009)
              + Partial Order Reduction

Full statespace search for:
              never claim         - (not selected)
              assertion violations - (disabled by -A flag)
              cycle checks        - (disabled by -DSAFETY)
              invalid end states +

State-vector 80 byte, depth reached 60, errors: 0
   27669 states, stored
   22864 states, matched
   50533 transitions (= stored+matched)
      4 atomic steps
hash conflicts:       301 (resolved)

Stats on memory usage (in Megabytes):
   2.428    equivalent memory usage for states (stored*(State-vector + overhead))
   2.488    actual memory usage for states (unsuccessful compression: 102.47%)
              state-vector as stored = 82 byte + 12 byte overhead
   2.000    memory used for hash table (-w19)
   0.305    memory used for DFS stack (-m10000)
   4.747    total actual memory usage

unreached in proctype requester
              (0 of 229 states)
unreached in proctype responder
              (0 of 163 states)
unreached in proctype BEreq
              (0 of 36 states)
unreached in proctype BEresp
              (0 of 26 states)
unreached in proctype recAckService
              (0 of 8 states)
unreached in proctype :init:
              (0 of 7 states)

pan: elapsed time 0.141 seconds
pan: rate 196234.04 states/second
pan: avg transition delay 2.7903e-006 usec
```

Figure D.7.: Verification Result for Invalid End States Analysis (Screenshot XSPIN)

`ing failed - stay in the same state */ inLoop: skip;` are executed. On the other hand, receiving an *AcceptanceAcknowledgement* is possible in every iteration of the non-progress cycle so that *strong fairness* is sufficient.

As SPIN does not support strongly fair process executions automatically, the *strong fairness* property must be encoded into the properties to be verified. Therefore two cases must be distinguished:

The first case is that the `do` loop of the *requester backend* process is performed exactly once, that means canceling the BT execution is never tried. The second case is that the `do` loop of the *requester backend* process is performed at least twice but at most finitely often, which means that canceling the BT execution is tried in vain at least once but at most finitely often. In order to distinguish between the two cases, the label `inLoop:` is inserted into line 27 of listing D.6 and used as follows.

If the requester backend's `proctype` never passes by this label in a particular system run then the `do` loop is executed exactly once (first case). Assume that the expression in corresponds to reaching the label `inLoop`. Then, the LTL formula $\Box ! in$ can be used to capture those system runs in which `inLoop` is never reached. The syntax for expressing that a label is reached in Promela is `processName[processID]@label`. As there is exactly one requester backend process for each system validation run, the expression `BEreq@inLoop` is sufficient for the process under consideration. Hence, $\Box ! BEreq@inLoop$ can be used to qualify the system runs that correspond to the first case. As the first case excludes non-progress cycles, strong fairness is not needed.

On the contrary, the second case enters the non-progress cycle so that strong fairness is needed. There is no limitation on the number of cancellation attempts defined in the BT execution model and hence the backend indeed may try to cancel the BT execution in vain infinitely often. However, it may be assumed that sooner or later the requester backend process will deliver either an exception or an *AcceptanceAcknowledgement* to its backend. From the perspective of SPIN, this assumption can be interpreted as the fact that the delivery of an exception or *AcceptanceAcknowledgement* which is enabled infinitely often is eventually performed. A general expression of strong fairness in LTL is $((\Box \Diamond P@U) \rightarrow (\Box \Diamond P@L))$ (cf. [62, page 141]). This expression says that if the label U of process P is reached once or infinitely often then label L of process P also must be reachable once or infinitely often. In order to transfer this expression to the second case, the additional label `passedLoop` is inserted into row 40 of listing D.6. Reaching this label corresponds to leaving the non-progress cycle of the requester backend which can be done by executing any branch of the `do` loop except for the `::true` branch. Based on the two labels `inLoop` and `passedLoop` the strong fairness property then can be expressed as $((\Box \Diamond in) \rightarrow (\Box \Diamond passed))$ if in corresponds to reaching `inLoop` and *passed* corresponds to reaching `passedLoop`.

Now, as the two formulas $\Box ! in$ and $((\Box \Diamond in) \rightarrow (\Box \Diamond passed))$ can be used to qualify the above two cases, these can be used to restrict the state space of the BT execution model to runs that do not execute the `do` loop of the *requester backend* infinitely often. One option to do so is using these formulas as preconditions for the actual property to be verified in logical implications.

Having defined how to exclude the non-progress cycle of the requester backend from the analysis, the actual BT execution model property to be verified, i.e. *state alignment*, is discussed next. The simulation runs of section D.3 show that reaching a *Success* state is possible for both control processes and both backends at the same time. So, technically speaking, the *state alignment* property of the BT execution model can be expressed as either the two control processes and the two backend processes all reach a *Failure* state or they all reach a *Success* state. The analysis of this property is split up into two parts. At first, correct termination of each individual control or backend process is analyzed. Then, the alignment of the process end states is verified.

For verifying correct termination of each backend and control process, the process state variables defined in the Promela models are used (`processName`EndstateFailure and `processName`EndstateSuccess). Initially, the two variables that are defined for a particular process are set to *false*. If correctly modified, both variables never carry a true value at the same time. In addition, at least one of the two variables must be true at the end of the process. In order to formally express these conditions in an LTL formula, let `p = processNameEndstateFailure` and `q = processNameEndstateSuccess` be the boolean variables representing failure and success of the respective process. Then the formula $(p \wedge !q) \vee (!p \wedge q)$ expresses that the process under consideration is either in a failure state or in a success state. However, the formula does not take into consideration that the process is neither in a success state nor in a failure state at the beginning. Put the other way around, the formula can only be required for the end state of the process. In order to formalize this fact, the LTL *"stability"* pattern can be used (cf. [62, pages 136-137]) that says that a particular property r must permanently hold true from some point in time onwards (formalized as $\Diamond\Box r$, cf. [62, page 137]). So, correct termination of a particular process can be expressed in LTL as

$$\Diamond\Box((p \wedge !q) \vee (!p \wedge q))$$

Listing D.10 shows the application of this formula to the control processes and backend processes of the BT execution model as well as the restriction to system runs that never try to cancel the BT execution from within the requester backend or do so at most finitely often. The formulas in lines 5 and 8 using the various variable definitions of lines 11-29 have to be verified by SPIN for ensuring correct termination of all control and backend processes.

Figure D.8 shows the exemplary input for verifying correct termination of the requester backend process under the condition of finitely often repetition of the requester backend's `do` loop (cancellation attempts of the BT execution). The code shown in the *"Never Claim"* area of the LTL editor is generated automatically from the formula and variable input. Moreover, the verification configuration options are shown in the small application window in the forefront of figure D.8. Finally, the result of the verification is shown (see marking in figure D.8). It is noteworthy that figure D.8 only shows the result for one formula and one backend process. The additional cases of checking the formulas of lines 5 and 8 against the other backend process and control processes have all been successfully verified as well.

Hence the claim can be established that each control and backend process terminates correctly from a local perspective. The alignment of the local results is analyzed next.

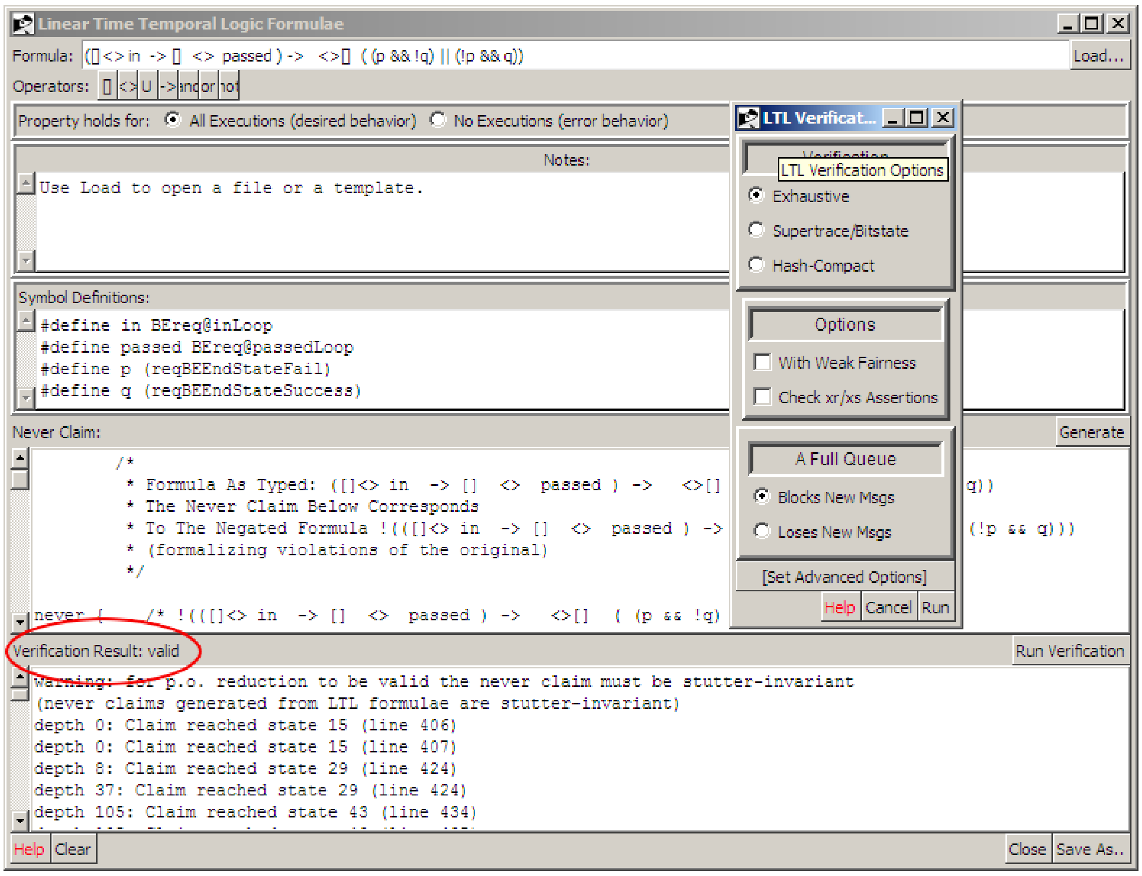
Listing D.10: LTL Formulas for Validating the Consistent Use of Local Process State Variables

```
/* Base LTL formula capturing ‘either Success or Failure’ */
<>[]( (p && !q) || (!p && q))

/* Base LTL formula for all runs that do not pass through
   the label ‘inLoop’ */
[] ! in -> <> [] ((p && !q) || (!p && q))

/* Base LTL formula for all runs that do pass through the
   label ‘inLoop’ */
([] <> in -> [] <> passed ) -> <> [] ((p && !q) || (!p && q)
   )

/* Definitions for: ‘in’ and ‘passed’ */
#define in BEreq@inLoop
#define passed BEreq@passedLoop

/* Redefinitions of p and q */
/* Requester process */
#define p (reqEndStateFail)
#define q (reqEndStateSuccess)

/* Responder process */
#define p (respEndStateFail)
#define q (respEndStateSuccess)

/* Requester backend */
#define p (reqBEEndStateFail)
#define q (reqBEEndStateSuccess)

/* Responder backend */
#define p (respBEEndStateFail)
#define q (respBEEndStateSuccess)
```

The *state alignment* property of the BT execution model, that means that the two control processes and the two backend processes all reach a *Failure* state or they all reach a *Success* state, does not express that all relevant processes reach an end state at the same time. However, once any of the participating processes makes a decision about success or failure (informally property p) then all other processes eventually have to make the same decision (informally property q). This can be formalized as follows in an abstract manner:

$$\Box\,(p \rightarrow \Diamond\Box q)$$

For the BT execution model, p and q have to concretized as follows:
p is either the disjunction of all process state variables signifying *Success* or the



Formula: ([]<> in -> [] <> passed) -> <>[] ((p && !q) || (!p && q))

Property holds for: [✓ All Executions (desired behavior)] No Executions (error behavior)

Notes:

```
Use Load to open a file or a template.
```

Symbol Definitions:

```
#define in BEreq@inLoop
#define passed BEreq@passedLoop
#define p (reqBEEndStateFail)
#define q (reqBEEndStateSuccess)
```

Never Claim:

```
        /*
         * Formula As Typed: ([]<> in  -> []  <>  passed ) ->  <>[]         q))
         * The Never Claim Below Corresponds
         * To The Negated Formula !(([]<> in  -> []  <>  passed ) ->        (!p && q)))
         * (formalizing violations of the original)
         */

never {    /* !(([]<> in  -> []  <>  passed ) ->  <>[]  ( (p && !q)
```

LTL Verification Options: [✓ Exhaustive] Supertrace/Bitstate Hash-Compact

Options: With Weak Fairness ☐ Check xr/xs Assertions ☐

A Full Queue: [✓ Blocks New Msgs] Loses New Msgs

Verification Result: valid

```
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 0: Claim reached state 15 (line 406)
depth 0: Claim reached state 15 (line 407)
depth 8: Claim reached state 29 (line 424)
depth 37: Claim reached state 29 (line 424)
depth 105: Claim reached state 43 (line 434)
```

Figure D.8.: Verification Result for the Local Process State Variables (XSPIN LTL Editor Screenshot)

disjunction of all process state variables signifying *Failure*. q then either is the conjunction of all process state variables signifying *Success* or the conjunction of all process state variables signifying *Failure* correspondingly.

The attentive reader may have noticed that verifying this formula does not ensure that any process eventually makes a decision. If no decision is made then the precondition of the implication ($p \rightarrow \Diamond\Box q$) does not hold and the overall implication is true. However, the first analysis step above already shows that each process eventually makes a decision. Hence, the precondition of this implication is true.

Listing D.11 shows the formalization of the second step in analyzing the *state alignment* property of the BT execution model. Again, infinite repetitions of the requester backend's `do` loop must be ruled out so that the formulas to be verified are given in lines 5 and 8. The definitions of the variables of these formulas are given in lines 11-32. As the verification of all combinations of formulas and variable definitions can be successfully verified (using the verification settings depicted in figure D.8) the claim can be established that the BT execution model ensures *state alignment* across the participating processes.

Listing D.11: LTL Formulas for Validating State Alignment

```
/* Base LTL formula representing consistent result across
   participants */
[] (p -> <> [] q)

/* Base LTL formula for all runs that do not pass through
   the label 'inLoop'  */
([]  ! in ->  [] (p -> <> [] q))

/* Base LTL formula for all runs that do pass through the
   label 'inLoop'  */
(([]<> in -> [] <> passed ) ->  [] (p -> <> [] q))

/* Definitions for: 'in' and 'passed' */
#define in BEreq@inLoop
#define passed BEreq@passedLoop

/* Redefinition of p and q for the 'Failure' case */
#define p (reqEndStateFail==true) ||
          (reqBEEndStateFail==true) ||
          (respEndStateFail==true) ||
          (respBEEndStateFail==true)
#define q (reqEndStateFail==true) &&
          (reqBEEndStateFail==true) &&
          (respEndStateFail==true) &&
          (respBEEndStateFail==true)

/* Redefinition of p and q for the 'Success' case */
#define p (reqEndStateSuccess==true) ||
          (reqBEEndStateSuccess==true) ||
          (respEndStateSuccess==true) ||
          (respBEEndStateSuccess==true)
#define q (reqEndStateSuccess==true) &&
          (reqBEEndStateSuccess==true) &&
          (respEndStateSuccess==true) &&
          (respBEEndStateSuccess==true)
```

D.5. Validation Results

In conclusion, this appendix shows that the BT execution model can adequately be represented as Promela processes. Based on this representation, XSPIN functionality can be used to explore the behavior of the BT execution model by means of simulation. In addition, the intended properties, that means *termination* and *state alignment*, can be proved. In so far, the execution model outperforms the guidelines of B2Bi standards (cf. chapter 4) and related scientific approaches (cf. chapter 7). However, it is vital to note that the properties proved only hold true for the abstract model that has been verified. As this abstract model has been created manually, *termination* and *state alignment* can only be taken for granted for the BPEL implementations of chapter 5 if no modifications of the abstract model's control flow are implied by the implementation process. Finally, it is noteworthy that the BT execution model could be verified at all by means of model checking considering the number of states and transitions of the corresponding state machines. This is largely due to the assumption of synchronous communication that eliminates the need for representing message buffers in the state space of the system under verification.

Bibliography

[1] H. J. Ahn, P. Childerhouse, G. Vossen, and H. Lee, “Rethinking XML-enabled agile supply chains,” *International Journal of Information Management*, vol. article in press, 2011.

[2] Z. Ammarguellat, “A control-flow normalization algorithm and its complexity,” *IEEE Trans. Softw. Eng.*, vol. 18, pp. 237–251, March 1992. [Online]. Available: http://dl.acm.org/citation.cfm?id=129809.129815

[3] S. Androutsellis-Theotokis, D. Spinellis, and V. Karakoidas, “Performing peer-to-peer e-business transactions: a requirements analysis and preliminary design proposal,” in *Proceedings of the IADIS International Conference on e-Commerce 2004*, N. Karmakar and P. Isaías, Eds., 2004, pp. 399–404.

[4] M. Backes, S. Moedersheim, B. Pfitzmann, and L. Vigano, “Symbolic and cryptographic analysis of the secure WS-ReliableMessaging scenario,” in *Proceedings of Foundations of Software Science and Computational Structures (FOSSACS), Vienna, Austria*, ser. Lecture Notes in Computer Science, vol. 3921. Springer, March 2006, pp. 428–445.

[5] K. Baïna, B. Benatallah, F. Casati, and F. Toumani, “Model-driven web service development,” in *Proceedings of the 16th International Conference on Advanced Information Systems Engineering, CAiSE 2004, Riga, Latvia*, ser. Lecture Notes in Computer Science, A. Persson and J. Stirna, Eds. Springer Berlin / Heidelberg, June 2004, vol. 3084, pp. 527–543. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-25975-6_22

[6] A. Barbir, C. Hobbs, E. Bertino, F. Hirsch, and L. Martino, “Challenges of testing Web Services and security in SOA implementations,” in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds. Springer Berlin Heidelberg, 2007, pp. 395–440.

[7] A. Barker, P. Besana, D. Robertson, and J. B. Weissman, “The benefits of service choreography for data-intensive computing,” in *CLADE ’09: Proceedings of the 7th international workshop on Challenges of large applications in distributed environments.* New York, NY, USA: ACM, 2009, pp. 1–10.

[8] A. Barker, C. D. Walton, and D. Robertson, “Choreographing web services,” *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 152–166, 2009.

[9] A. P. Barros and A. H. M. ter Hofstede, "Towards the construction of workflow-suitable conceptual modelling techniques," *Information Systems Journal*, vol. 8, no. 4, pp. 313–337, October 1998.

[10] A. P. Barros, G. Decker, and M. Dumas, "Multi-staged and multi-viewpoint service choreography modelling," in *Proceedings of the Workshop on Software Engineering Methods for Service Oriented Architecture (SEMSOA), Hannover, Germany, May 10-11, 2007*, ser. CEUR Workshop Proceedings, vol. 244, May 2007.

[11] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Service interaction patterns," in *Proceedings of the 3rd International Conference on Business Process Management (BPM), Nancy, France.* Springer Verlag, 2005, pp. 302–318.

[12] *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*, BEA Systems, IBM Corporation, Microsoft Corporation Inc., TIBCO Software Inc., February 2005, first specification (Version 1.0). [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rm/ws-reliablemessaging200502.pdf

[13] J. Becker, *Handelsinformationssysteme.* Landsberg/Lech: Verl. Moderne Industrie, 1996.

[14] D. Beimborn, S. Mintert, and T. Weitzel, "Web services und ebXML," *Wirtschaftsinformatik*, vol. 44, no. 3, pp. 277–280, 2002.

[15] B. Benatallah, F. Casati, and F. Toumani, "Representing, analysing and managing web service protocols," *Data Knowl. Eng.*, vol. 58, no. 3, pp. 327–357, 2006.

[16] T. Benker, S. Fritzemeier, M. Geiger, S. Harrer, T. Kessner, J. Schwalb, A. Schönberger, and G. Wirtz, "QoS-enabled B2B integration," Otto-Friedrich-Universität Bamberg, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik 80, May 2009. [Online]. Available: http://www.opus-bayern.de/uni-bamberg/volltexte/2009/205/

[17] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and Software Verification : Model-Checking Techniques and Tools*, 1st ed. Berlin: Springer-Verlag, August 2001.

[18] P. Besana and A. Barker, "An executable calculus for service choreography," in *Proceedings of On the Move 2009 Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Vilamoura, Portugal*, ser. Lecture Notes in Computer Science, R. Meersman, T. Dillon, and P. Herrero, Eds. Springer Berlin / Heidelberg, 2009, vol. 5870, pp. 373–380. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-05148-7_26

[19] K. Bhargavan, R. Corin, C. Fournet, and A. D. Gordon, "Secure sessions for web services," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 2, pp. article no. 8, 46 pages, 2007.

[20] D. Bianchini, C. Cappiello, V. D. Antonellis, and B. Pernici, "P2S: A methodology to enable inter-organizational process design through web services," in *Proceedings of the 21st International Conference on Advanced Information Systems Engineering, CAiSE 2009, Amsterdam, The Netherlands*, ser. Lecture Notes in Computer Science, P. van Eck, J. Gordijn, and R. Wieringa, Eds. Springer Berlin / Heidelberg, June 2009, vol. 5565, pp. 334–348. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02144-2_28

[21] A. Brogi and R. Popescu, "Automated generation of BPEL adapters," in *Proceedings of the 4th International Conference on Service-Oriented Computing - ICSOC 2006, Chicago, IL, USA*, ser. Lecture Notes in Computer Science, A. Dan and W. Lamersdorf, Eds. Springer Berlin / Heidelberg, december 2006, vol. 4294, pp. 27–39. [Online]. Available: http://dx.doi.org/10.1007/11948148_3

[22] G. Bruno, "Modeling and using business collaborations," in *Proceedings of the 1st Int. Conf. on Interoperability of enterprise software and applications INTEROP-ESA 2005, Geneva, Switzerland*, D. Konstantas, J.-P. Bourrières, M. Léonard, and N. Boudjlida, Eds. Springer London, 2005, pp. 114–125. [Online]. Available: http://dx.doi.org/10.1007/1-84628-152-0_11

[23] T. Bultan and X. Fu, "Specification of realizable service conversations using collaboration diagrams," *Service Oriented Computing and Applications, Springer*, vol. 2, no. 1, pp. 27–39, 2008.

[24] T. Bultan, J. Su, and X. Fu, "Analyzing conversations of web services," *IEEE Internet Computing*, vol. 10, no. 1, pp. 18–25, 2006.

[25] E. M. Clarke Jr., O. Grumberg, and D. A. Peled, *Model checking.* Cambridge, MA, USA: MIT Press, 1999.

[26] P. Dadam and M. Reichert, "The ADEPT project: a decade of research and development for robust and flexible process support," *Computer Science - Research and Development*, vol. 23, no. 2, pp. 81–97, 2009.

[27] G. Decker and A. P. Barros, "Interaction modeling using BPMN," in *Proceedings of the 1st International Workshop on Collaborative Business Processes (CBP), Brisbane, Australia*, ser. LNCS, no. 4928. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 208–219.

[28] G. Decker, A. P. Barros, F. M. Kraft, and N. Lohmann, "Non-desynchronizable service choreographies," in *ICSOC '08: Proceedings of the 6th International Conference on Service-Oriented Computing.* Berlin, Heidelberg: Springer-Verlag, 2008, pp. 331–346.

[29] G. Decker, O. Kopp, and A. P. Barros, “An introduction to service choreographies,” *Information Technology*, vol. 50, no. 2, pp. 122–127, 2008.

[30] G. Decker, O. Kopp, F. Leymann, K. Pfitzner, and M. Weske, “Modeling service choreographies using BPMN and BPEL4Chor,” in *CAiSE ’08: Proceedings of the 20th international conference on Advanced Information Systems Engineering, Montpellier, France.* Berlin, Heidelberg: Springer-Verlag, 2008, pp. 79–93.

[31] G. Decker, O. Kopp, F. Leymann, and M. Weske, “BPEL4Chor: Extending BPEL for modeling choreographies,” in *Proceedings of the 2007 IEEE International Conference on Web Services (ICWS), July 9-13, 2007, Salt Lake City, Utah, USA*, 2007, pp. 296–303.

[32] ——, “Interacting services: From specification to execution,” *Data & Knowledge Engineering*, vol. 68, no. 10, pp. 946 – 972, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/B6TYX-4W4JDJH-1/2/ac04033b7102e246c41244b0efc79812

[33] G. Decker and M. Weske, “Local enforceability in interaction petri nets,” in *Proceedings of the 5th International Conference on Business Process Management (BPM 2007), Brisbane, Australia, September 24-28*, 2007, pp. 305–319.

[34] P. Derler and R. Weinreich, “Models and tools for SOA governance,” in *Proc. of the 2nd International Conference on Trends in Enterprise Application Architecture (TEAA 2006)*, Berlin, Germany, 2006, pp. 112–126.

[35] R. Dijkman and M. Dumas, “Service-oriented Design: A Multi-viewpoint Approach,” *International Journal of Cooperative Information Systems*, vol. 13, no. 4, pp. 337–368, 2004.

[36] A. Dogac, Y. Tambag, P. Pembecioglu, S. Pektas, G. Laleci, G. Kurt, S. Toprak, and Y. Kabak, “An ebXML infrastructure implementation through UDDI registries and RosettaNet PIPs,” in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, ser. SIGMOD’02. New York, NY, USA: ACM, 2002, pp. 512–523. [Online]. Available: http://doi.acm.org/10.1145/564691.564750

[37] J. Dorn, C. Grün, H. Werthner, and M. Zapletal, “A survey of B2B methodologies and technologies: From business models towards deployment artifacts,” in *HICSS ’07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences, Waikoloa, Big Island, Hawaii, USA.* IEEE Computer Society, 2007.

[38] R. Eshuis, “Reconciling statechart semantics,” *Sci. Comput. Program.*, vol. 74, no. 3, pp. 65–99, 2009.

[39] R. Eshuis, P. Brimont, E. Dubois, B. Grégoire, and S. Ramel, “Animating ebXML transactions with a workflow engine,” in *Proceedings of the OTM Confederated International Conferences, On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, Catania, Sicily, Italy*, November 2003, pp. 426–443.

[40] R. Eshuis and P. W. P. J. Grefen, “Composing services into structured processes.” *International Journal of Cooperative Information Systems*, vol. 18, no. 2, pp. 309–337, 2009.

[41] O. K. Ferstl and E. J. Sinz, “Modeling of business systems using SOM,” in *Handbook on Architectures of Information Systems*, ser. International Handbooks on Information Systems, P. Bernus, K. Mertins, and G. Schmidt, Eds. Springer Berlin Heidelberg, 2006, pp. 347–367.

[42] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: http://portal.acm.org/citation.cfm?id=932295

[43] D. Florescu, A. Grünhagen, and D. Kossmann, “XL: an XML programming language for web service specification and composition,” in *Proceedings of the 11th international conference on World Wide Web*, ser. WWW ’02. New York, NY, USA: ACM, 2002, pp. 65–76. [Online]. Available: http://doi.acm.org/10.1145/511446.511456

[44] K. Gavrylyuk, O. Hrebicek, and S. Batres. (2005) WCF (Indigo) Interoperability Lab: Reliable Messaging. Word Document (.doc). Microsoft. [Online]. Available: http://mssoapinterop.org/ilab/RM/WCFInteropPlugFest_RM.doc

[45] M. Geiger, A. Schönberger, and G. Wirtz, “A proposal for checking the conformance of ebBP-ST choreographies and WS-BPEL orchestrations,” in *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS), Karlsruhe, Germany, February 21-22, 2011*, ser. CEUR Workshop Proceedings. CEUR-WS.org, Feb 2011, pp. 24–25.

[46] ——, “Towards automated conformance checking of ebBP-ST choreographies and corresponding WS-BPEL based orchestrations,” in *Proceedings of 2011 Conf. on Software Engineering and Knowledge Engineering (SEKE’2011), Miami, Florida, USA.* Knowledge Systems Institute, 7.-9. July 2011.

[47] G. M. Giaglis, R. J. Paul, and G. I. Doukidis, “Simulation for intra- and inter-organisational business process modelling,” in *WSC ’96: Proceedings of the 28th conference on Winter simulation.* Washington, DC, USA: IEEE Computer Society, 1996, pp. 1297–1304.

[48] F. Goethals, J. Vandenbulcke, W. Lemahieu, M. Snoeck, and B. Cumps, "Two basic types of business-to-business integration," *International Journal of E-Business Research (IJEBR)*, vol. 1, no. 1, pp. 1–15, 2005.

[49] F. G. Goethals, "Important issues for evaluating inter-organizational data integration configurations," *The Electronic Journal Information Systems Evaluation*, vol. 11, no. 3, pp. 185–196, 2008. [Online]. Available: http://www.ejise.com/volume-11/volume11-issue3/Goethals.pdf

[50] S. Gregor and D. Jones, "The anatomy of a design theory," *Journal of the Association for Information Systems*, vol. 8, no. 5, pp. 312–335, 2007.

[51] A. Gunasekaran and E. W. T. Ngai, "Information systems in supply chain integration and management," *European Journal of Operational Research*, vol. 159, no. 2, pp. 269–295, December 2004. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v159y2004i2p269-295.html

[52] A. Gunasekaran, C. Patel, and R. E. McGaughey, "A framework for supply chain performance measurement," *International Journal of Production Economics*, vol. 87, no. 3, pp. 333–347, February 2004. [Online]. Available: http://ideas.repec.org/a/eee/proeco/v87y2004i3p333-347.html

[53] R. Hamadi and B. Benatallah, "A petri net-based model for web service composition," in *CRPITS'17: Proceedings of the Fourteenth Australasian database conference on Database technologies 2003*. Darlinghurst, Australia: Australian Computer Society, Inc., 2003, pp. 191–200.

[54] S. Harrer, A. Schönberger, and G. Wirtz, "A model-driven approach for monitoring ebBP BusinessTransactions," in *Proceedings of the 7th World Congress on Services 2011(SERVICES2011), Washington, D.C., USA*. IEEE, July 2011.

[55] R. Hauser and J. Koehler, "Compiling process graphs into executable code," in *Proceedings of the Third International Conference on Generative Programming and Component Engineering, GPCE 2004, Vancouver, Canada*, ser. Lecture Notes in Computer Science, G. Karsai and E. Visser, Eds. Springer Berlin / Heidelberg, October 2004, vol. 3286, pp. 129–243. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30175-2_17

[56] T. Hettel, C. Flender, and A. P. Barros, "Scaling choreography modelling for B2B value-chain analysis," in *BPM '08 Proceedings of the 6th International Conference on Business Process Management, Milan, Italy*, ser. Lecture Notes in Computer Science, M. Dumas, M. Reichert, and M.-C. Shan, Eds. Springer Berlin / Heidelberg, 2008, vol. 5240, pp. 294–309. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85758-7_22

[57] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[58] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, pp. 666–677, August 1978. [Online]. Available: http://doi.acm.org/10.1145/359576.359585

[59] B. Hofreiter and C. Huemer, "A model-driven top-down approach to inter-organizational systems: From global choreography models to executable BPEL," in *Joint Conference on E-Commerce Technology (CEC'08) and Enterprise Computing, E-Commerce, and E-Services (EEE'08).* Crystal City, Washington D.C., USA: IEEE, 7 2008.

[60] B. Hofreiter, C. Huemer, and J.-H. Kim, "Choreography of ebXML business collaborations," *Information Systems and E-Business Management*, vol. 4, pp. 221–243, 2006, 10.1007/s10257-005-0016-3. [Online]. Available: http://dx.doi.org/10.1007/s10257-005-0016-3

[61] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal, "Deriving executable BPEL from UMM business transactions," in *Proceedings of the IEEE International Conference on Services Computing (SCC), Salt Lake City, UT, USA*, 2007, pp. 178–186.

[62] G. J. Holzmann, *The SPIN Model Checker.* Addison-Wesley Pearson Education, September 2003.

[63] C. Huemer, P. Liegl, R. Schuster, H. Werthner, and M. Zapletal, "Inter-organizational systems: From business values over business processes to deployment," in *Proceedings of the 2nd International IEEE Conference on Digital Ecosystems and Technologies (DEST2008), Phitsanulok, Thailand.* IEEE, 2008.

[64] C. Huemer, P. Liegl, R. Schuster, and M. Zapletal, "B2B Services: Worksheet-Driven Development of Modeling Artifacts and Code," *The Computer Journal*, vol. 52, no. 8, pp. 1006–1026, 2009. [Online]. Available: http://comjnl.oxfordjournals.org/content/52/8/1006.abstract

[65] C. Huemer and M. Zapletal, "A state machine executing UMM business transactions," in *Proceedings of the 2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007)*, IEEE Computer Society. Cairns (Australia): IEEE Computer Society, 2007. [Online]. Available: http://publik.tuwien.ac.at/files/pub-inf_4580.pdf

[66] IETF, *ODETTE File Transfer Protocol 2*, IETF, 11 2007. [Online]. Available: http://tools.ietf.org/html/rfc5024

[67] M. Ilger and M. Zapletal, “An implementation to transform business collaboration models to executable process specifications,” in *Proceedings of the conference on Service-Oriented Electronic Commerce at the Multikonferenz Wirtschaftsinformatik 2006, Passau, Germany*, M. Schoop, C. Huemer, M. Rebstock, and M. Bichler, Eds. GI-Edition - Lecture Notes in Informatics (LNI), 2006, pp. 9–23.

[68] M. Indulska, J. Recker, M. Rosemann, and P. Green, “Business process modeling: Current issues and future challenges,” in *CAiSE ’09: Proceedings of the 21st International Conference on Advanced Information Systems Engineering.* Berlin, Heidelberg: Springer-Verlag, 2009, pp. 501–514.

[69] ISO/IEC, *Information technology - Open-edi reference model*, 2nd ed., ISO/IEC, May 2004. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c037354_ISO_IEC_14662_2004(E).zip

[70] ——, *ISO/IEC 15909-1:2004: Software and system engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation*, ISO/IEC, December 2004. [Online]. Available: http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=38225&COMMID=&scopelist=

[71] J. Jung, W. Hur, S.-H. Kang, and H. Kim, “Business process choreography for B2B collaboration,” *IEEE Internet Computing*, vol. 8, pp. 37–45, January 2004.

[72] R. Kazhamiakin and M. Pistore, “Choreography conformance analysis: Asynchronous communications and information alignment,” in *Proceedings of the Third International Workshop on Web Services and Formal Methods, WS-FM 2006 Vienna, Austria*, ser. Lecture Notes in Computer Science, M. Bravetti, M. Nunez, and G. Zavattaro, Eds. Springer Berlin / Heidelberg, September 2006, vol. 4184, pp. 227–241.

[73] R. Khalaf, A. Keller, and F. Leymann, “Business processes for web services: principles and applications,” *IBM Syst. J.*, vol. 45, pp. 425–446, January 2006. [Online]. Available: http://dx.doi.org/10.1147/sj.452.0425

[74] R. Khalaf, “From RosettaNet PIPs to BPEL processes: A three level approach for business protocols,” in *Proceedings of the 3rd International Conference on Business Process Management, BPM 2005, Nancy, France*, ser. Lecture Notes in Computer Science, W. M. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, Eds. Springer Berlin / Heidelberg, September 2005, vol. 3649, pp. 364–373. [Online]. Available: http://dx.doi.org/10.1007/11538394_25

[75] ——, “From RosettaNet PIPs to BPEL processes: A three level approach for business protocols,” *Data & Knowledge Engineering*, vol. 61, no. 1, pp. 23–38, 2007.

[76] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler, "On structured workflow modelling," in *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering.* London, UK: Springer-Verlag, 2000, pp. 431–445.

[77] H. D. Kim, "BPMN-based modeling of B2B business processes from the neutral perspective of UMM/BPSS," in *Proceedings of the 2008 IEEE International Conference on e-Business Engineering, ICEBE, Xi'An, China.* Washington, DC, USA: IEEE Computer Society, October 2008, pp. 417–422. [Online]. Available: http://dl.acm.org/citation.cfm?id=1471605.1472141

[78] H. Kim, "Conceptual modeling and specification generation for B2B business processes based on ebXML," *SIGMOD Rec.*, vol. 31, pp. 37–42, March 2002. [Online]. Available: http://doi.acm.org/10.1145/507338.507346

[79] J.-H. Kim and C. Huemer, "From an ebXML BPSS choreography to a BPEL-based implementation," *SIGecom Exch.*, vol. 5, no. 2, pp. 1–11, 2004.

[80] J. Koehler and R. Hauser, "Untangling unstructured cyclic flows - a solution based on continuations," in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds. Springer Berlin / Heidelberg, 2004, vol. 3290, pp. 121–138. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30468-5_10

[81] O. Kopp, D. Martin, D. Wutke, and F. Leymann, "The difference between graph-based and block-structured business process modelling languages," *Enterprise Modelling and Information Systems Architectures*, vol. 4, no. 1, pp. 3–13, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/emisaij/emisaij4.html#KoppMWL09

[82] O. Kopp, B. Wetzstein, R. Mietzner, S. Pottinger, D. Karastoyanova, and F. Leymann, "A model-driven approach to implementing coordination protocols in BPEL," in *Proceedings of the 1st International Workshop on Model-Driven Engineering for Business Process Management (MDE4BPM 2008)*, ser. Lecture Notes in Business Information Processing, D. Ardagna, M. Mecella, J. Yang, W. M. van der Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, and C. Szyperski, Eds. Springer Berlin Heidelberg, September 2008, vol. 17, pp. 188–199. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00328-8_19

[83] O. Kopp, M. Wieland, and F. Leymann, "Towards Choreography Transactions," in *Proceedings of the 1st Central-European Workshop on Services and their Composition, ZEUS 2009, Stuttgart, Germany, March 2–3, 2009*, ser. CEUR Workshop Proceedings, O. Kopp and N. Lohmann, Eds., vol. 438. Stuttgart: CEUR-WS.org, 2009, pp.

49–54. [Online]. Available: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2009-28&engl=

[84] M. Kovács, D. Varró, and L. Gönczy, “Formal analysis of BPEL workflows with compensation by model checking,” *International Journal of Computer Systems Science and Engineering, Special Issue: Engineering Fault Tolerant Systems*, vol. 23, no. 5, pp. 349–363, Sep 2008.

[85] V. Kozyura, A. Roth, and W. Wei, “Local enforceability and inconsumable messages in choreography models,” in *Proceedings of 4th South-East European Workshop on Formal Methods (SEEFM’09),Thessaloniki, Greece*, 2009.

[86] G. Kramler, E. Kapsammer, W. Retschitzegger, and G. Kappel, “Towards using UML 2 for modelling web service collaboration protocols,” in *Proceedings of the 1st Int. Conf. on Interoperability of enterprise software and applications INTEROP-ESA 2005, Geneva, Switzerland*, D. Konstantas, J.-P. Bourrieres, M. Leonard, and N. Boudjlida, Eds. Springer London, 2005. [Online]. Available: http://dx.doi.org/10.1007/1-84628-152-0_21

[87] C. Kruse, *Referenzmodellgestütztes Geschäftsprozessmanagement.* Wiesbaden: Gabler, 1996.

[88] W. Lam, “Investigating success factors in enterprise application integration: a case-driven analysis,” *Eur. J. Inf. Syst.*, vol. 14, no. 2, pp. 175–187, 2005.

[89] W. Lam and V. Shankararaman, “An enterprise integration methodology,” *IT Professional*, vol. 6, no. 2, pp. 40–48, 2004.

[90] D. M. Lambert and M. C. Cooper, “Issues in supply chain management,” *Industrial Marketing Management*, vol. 29, no. 1, pp. 65 – 83, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/B6V69-3YDGKGP-8/2/6b65247be734b1607fc222cee08e2a99

[91] N. Laranjeiro and M. Vieira, “Deploying fault tolerant web service compositions,” *International Journal of Computer Systems Science and Engineering, Special Issue: Engineering Fault Tolerant Systems*, vol. 23, no. 5, pp. 337–348, Sep 2008.

[92] K. Lassen and W. M. P. van der Aalst, “WorkflowNet2BPEL4WS: A tool for translating unstructured workflow processes to readable BPEL,” in *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Montpellier, France*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds. Springer Berlin / Heidelberg, 2006, vol. 4275, pp. 127–144. [Online]. Available: http://dx.doi.org/10.1007/11914853_9

[93] J. Lenhard, A. Schönberger, and G. Wirtz, “Edit distance-based pattern support assessment of orchestration languages,” in *Proceedings of On the Move 2011*

Confederated International Conferences: CoopIS, IS, DOA and ODBASE, Hersonissos, Crete, Greece, Oct 19 - 21, 2011, ser. Lecture Notes in Computer Science. Springer, 2011.

[94] ——, "Streamlining pattern support assessment for service composition languages," in *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS), Karlsruhe, Germany, February 21-22, 2011*, ser. CEUR Workshop Proceedings. CEUR-WS.org, Feb 2011, pp. 112–119.

[95] P. Liegl, M. Zapletal, C. Pichler, and M. Strommer, "State-of-the-art in business document standards," in *Proceedings of the 8th IEEE International Conference on Industrial Informatics, Osaka, Japan*, 2010, pp. 1–8. [Online]. Available: http://publik.tuwien.ac.at/files/PubDat_186928.pdf

[96] A. Liu, L. Huang, Q. Li, and M. Xiao, "Fault-tolerant orchestration of transactional web services," in *Proceedings of the 7th International Conference on Web Information Systems Engineering, WISE 2006, Wuhan, China*, October 2006, pp. 90–101.

[97] R. Liu and A. Kumar, "An analysis and taxonomy of unstructured workflows," in *Proceedings of the 3rd International Conference on Business Process Management, BPM 2005, Nancy, France*, ser. Lecture Notes in Computer Science, W. M. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, Eds. Springer Berlin / Heidelberg, 2005, vol. 3649, pp. 268–284. [Online]. Available: http://dx.doi.org/10.1007/11538394_18

[98] N. Lohmann, O. Kopp, F. Leymann, and W. Reisig, "Analyzing BPEL4Chor: verification and participant synthesis," in *Proceedings of the 4th international conference on Web services and formal methods, Brisbane, Australia*, ser. WS-FM'07. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 46–60. [Online]. Available: http://dl.acm.org/citation.cfm?id=1791676.1791680

[99] N. Lohmann and K. Wolf, "Artifact-centric choreographies," in *Proceedings of the 8th International Conference on Service-Oriented Computing, ICSOC 2010, San Francisco, CA, USA*, ser. Lecture Notes in Computer Science, P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds. Springer Berlin / Heidelberg, December 2010, vol. 6470, pp. 32–46. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-17358-5_3

[100] R. Lu and S. W. Sadiq, "A survey of comparative business process modeling approaches," in *Proceedings of the 10th International Conference of Business Information Systems (BIS 2007)*, ser. Lecture Notes in Computer Science, vol. 4439. Poznan, Poland: Springer, April 2007, pp. 82–94.

[101] L. T. Ly, K. Göser, S. Rinderle-Ma, and P. Dadam, "Compliance of semantic constraints - a requirements analysis for process management systems," in *Proc.*

1st Int'l Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS'08), Montpellier, France, 2008.

[102] M. Madiesh and G. Wirtz, "A top-down method for B2B process design using SOA," in *Proceedings of the 2008 International Conference on Software Engineering Research & Practice, SERP 2008, July 14-17, 2008, Las Vegas Nevada, USA*, 2008, pp. 444–450.

[103] ——, "A top-down method for secure SOA-based B2B processes," in *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010), Redwood City, San Francisco Bay, CA, USA*, July 2010, pp. 698–703.

[104] M. L. Markus, A. Majchrzak, and L. Gasser, "A design theory for systems that support emergent knowledge processes," *MIS Quarterly*, pp. 179–212, 2002.

[105] A. Martens, "Consistency between executable and abstract processes," in *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE 2005, Hong Kong, China*, April 2005, pp. 60–67.

[106] L. Martino and E. Bertino, "Security for web services: Standards and research issues," *Int. J. Web Services Res.*, vol. 6, no. 4, pp. 48–74, 2009.

[107] S. McIlvenna, M. Dumas, and M. T. Wynn, "Synthesis of orchestrators from service choreographies." in *Proc. of the Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM)*, ser. CRPIT, M. Kirchberg and S. Link, Eds., vol. 96. Australian Computer Society, 2009, pp. 129–138. [Online]. Available: http://dblp.uni-trier.de/db/conf/apccm/apccm2009.html#McIlvennaDW09

[108] A. McNeile, "Protocol contracts with application to choreographed multiparty collaborations," *Service Oriented Computing and Applications*, vol. 4, pp. 109–136, 2010, 10.1007/s11761-010-0060-9. [Online]. Available: http://dx.doi.org/10.1007/s11761-010-0060-9

[109] B. Medjahed, B. Benatallah, A. Bouguettaya, A. Ngu, and A. Elmagarmid, "Business-to-business interactions: issues and enabling technologies," *The VLDB Journal*, vol. 12, no. 1, pp. 59–85, 2003.

[110] J. Mendling and M. Hafner, "From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL," *Proceedings of OTM 2005 Workshops, Agia Napa, Cyprus*, vol. LNCS 3762, pp. 506–515, 2005.

[111] J. Mendling, K. B. Lassen, and U. Zdun, "On the transformation of control flow between block-oriented and graph-oriented process modelling languages," *International Journal of Business Process Integration and Management*, vol. 3, no. 2, pp. 96 – 108, 2008.

[112] J. T. Mentzer, W. DeWitt, J. S. Keebler, S. Min, N. W. Nix, C. D. Smith, and Z. G. Zacharia, "Defining supply chain management," *JOURNAL OF BUSINESS LOGISTICS*, vol. 22, no. 2, pp. 1–26, 2001.

[113] R. Milner, "A calculus of communicating systems," vol. 92, 1980.

[114] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, i," *Information and Computation*, vol. 100, no. 1, pp. 1 – 40, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0890540192900084

[115] ——, "A calculus of mobile processes, ii," *Information and Computation*, vol. 100, no. 1, pp. 41 – 77, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0890540192900095

[116] D. Moberg and R. Drummond, *RFC 4130:MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2)*, The Internet Engineering Task Force (IETF), July 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4130.txt

[117] L. E. Moser, P. M. Melliar-Smith, and W. Zhao, "Building dependable and secure web services," *Journal of Software*, vol. 2, no. 1, pp. 14–26, 2007.

[118] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 815–824. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367607

[119] S. Mouzakitis and D. Askounis, "A capability assessment framework for the adoption of B2B integration systems," in *WSKS '08: Proceedings of the 1st world summit on The Knowledge Society.* Berlin, Heidelberg: Springer-Verlag, 2008, pp. 451–459.

[120] B. Mutschler, B. Weber, and M. Reichert, "Workflow management versus case handling: results from a controlled software experiment," in *Proceedings of the 2008 ACM symposium on Applied computing, Fortaleza, Brazil.* ACM, Mar 2008, pp. 82–89.

[121] K.-D. Naujok and C. Huemer, "Designing ebXML - the work of UN/CEFACT," in *Ontologies-Based Business Integration.* Heidelberg: Springer, 2008, pp. 79–93, eingeladen.

[122] H. Nezhad, B. Benatallah, F. Casati, and F. Toumani, "Web services interoperability specifications," *Computer*, vol. 39, no. 5, pp. 24–32, May 2006.

[123] *Northern European Subset Profiles*, 2nd ed., Northern European working group for development of a subset for UBL 2.0, July 2007. [Online]. Available: http://www.nesubl.eu

[124] J.-M. Nurmilaakso, “ICT solutions and labor productivity: evidence from firm-level data,” *Electronic Commerce Research*, vol. 9, no. 3, pp. 173–181, 2009.

[125] J. Nurmilaakso and P. Kotinurmi, “A review of XML-based supply-chain integration,” *Production Planning and Control*, vol. 15, no. 6, pp. 608–621, 2004.

[126] A. Nysetvold and J. Krogstie, “Assessing business processing modeling languages using a generic quality framework,” in *Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD‘05), held in conjunctiun with the 17th Conference on Advanced Information Systems 2005 (CAiSE 2005), Porto, Portugal*, T. Halpin, K. Siau, and J. Krogstie, Eds., 2005.

[127] OASIS, *ebXML Collaboration-Protocol Profile and Agreement Specification Version 3.0 (Editor’s Draft, latest state as of June 2009)*, OASIS. [Online]. Available: http://www.oasis-open.org/committees/download.php/31996/ebcppa-v3.0-Spec-wd-r01-en-pete4.pdf

[128] ——, *Collaboration-Protocol Profile and Agreement Specification Version 2.0*, OASIS, sep 2002. [Online]. Available: http://www.oasis-open.org/committees/ebxml-cppa/documents/ebcpp-2.0.pdf

[129] ——, *ebXML Message Service Specification*, 2nd ed., OASIS, April 2002. [Online]. Available: http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf

[130] OASIS, *UDDI Spec Technical Committee Specification*, 3rd ed., OASIS, 10 2003. [Online]. Available: http://uddi.org/pubs/uddi-v3.0.1-20031014.htm

[131] OASIS, *ebXML Registry Information Model Version 3.0*, OASIS, May 2005.

[132] ——, *ebXML Registry Services and Protocols Version 3.0*, OASIS, May 2005.

[133] OASIS, *UDDI version 3.0.2*, OASIS, February 2005. [Online]. Available: http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf

[134] OASIS, *ebXML Business Process Specification Schema Technical Specification*, 2nd ed., OASIS, December 2006. [Online]. Available: http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en.pdf

[135] ——, *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, OASIS, February 2006. [Online]. Available: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[136] ——, *ebXML Messaging Services Version 3.0: Part 1, Core Features*, OASIS, October 2007. [Online]. Available: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/os/ebms_core-3.0-spec-os.pdf

[137] ——, *Web Services Business Process Execution Language*, 2nd ed., April 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[138] OASIS, *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1*, OASIS, January 2008. [Online]. Available: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01-e1.pdf

[139] ——, *Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os.pdf

[140] ——, *Web Services Business Activity (WS-BusinessActivity) Version 1.2*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os.pdf

[141] OASIS, *Web Services Coordination (WS-Coordination) Version 1.2*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os.pdf

[142] ——, *Web Services Reliable Messaging Policy Assertion (WS-RM Policy) Version 1.2*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.2-spec-os.pdf

[143] OASIS, *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.pdf

[144] ——, *WS-SecureConversation 1.4*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf

[145] OASIS, *WS-SecurityPolicy 1.3*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf

[146] ——, *WS-Trust 1.4*, OASIS, February 2009. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf

[147] OASIS; UN/CEFACT, *ebXML Business Process Specification Schema*, 1st ed., OASIS; UN/CEFACT, May 2001. [Online]. Available: http://www.ebxml.org/specs/ebBPSS.pdf

[148] L. O'Brien, P. Merson, and L. Bass, "Quality attributes for service-oriented architectures," in *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments.* Washington, DC, USA: IEEE Computer Society, 2007, p. 3.

[149] OMG, *Unified Modeling Language (OMG UML), Superstructure*, 2nd ed., May 2010.

[150] ——, *Business Process Model and Notation, v2.0*, OMG, January 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[151] C. Ouyang, M. Dumas, S. Breutel, and A. ter Hofstede, "Translating standard process models to BPEL," in *Proceedings of the 18th International Conference on Advanced Information Systems Engineering, CAiSE 2006, Luxembourg, Luxembourg*, ser. Lecture Notes in Computer Science, E. Dubois and K. Pohl, Eds. Springer Berlin / Heidelberg, June 2006, vol. 4001, pp. 417–432. [Online]. Available: http://dx.doi.org/10.1007/11767138_28

[152] C. Ouyang, M. Dumas, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "From BPMN process models to BPEL web services," in *ICWS '06: Proceedings of the IEEE International Conference on Web Services.* Washington, DC, USA: IEEE Computer Society, 2006, pp. 285–292.

[153] C. Ouyang, M. Dumas, A. H. ter Hofstede, and W. M. van der Aalst, "Pattern-based translation of BPMN process models to BPEL web services," *International Journal of Web Services Research (JWSR)*, vol. 5, no. 1, pp. 42–62, 2007. [Online]. Available: http://eprints.qut.edu.au/6810/

[154] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling, "From business process models to process-oriented software systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 19, pp. 2:1–2:37, August 2009. [Online]. Available: http://doi.acm.org/10.1145/1555392.1555395

[155] M. P. Papazoglou, "Web services and business transactions," *World Wide Web: Internet and Web Information Systems*, vol. 6, pp. 49–91, March 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=634729.634760

[156] L. C. Paulson, "Inductive analysis of the internet protocol TLS," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 3, pp. 332–351, August 1999. [Online]. Available: http://doi.acm.org/10.1145/322510.322530

[157] C. Pautasso and G. Alonso, "JOpera: a toolkit for efficient visual composition of web services," *International Journal of Electronic Commerce*, vol. 9, no. 2, pp. 107–141, 2004.

[158] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services: making the right architectural decision," in *Proceeding*

of the 17th international conference on World Wide Web, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 805–814. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367606

[159] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[160] C. Pflügler, A. Schönberger, and G. Wirtz, "Introducing partner shared states into ebBP to WS-BPEL translations," in *Proc. iiWAS2009, 11th International Conference on Information Integration and Web-based Applications & Services, 14.-16. December 2009, Kuala Lumpur, Malaysia.* ACM, December 2009.

[161] K. Phalp and M. Shepperd, "Quantitative analysis of static models of processes," *Journal of Systems and Software*, vol. 52, no. 2-3, pp. 105 – 112, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0N-40962F9-3/2/025532797e57568a53472a442269e355

[162] K. T. Phalp, "The CAP framework for business process modelling," *Information and Software Technology*, vol. 40, no. 13, pp. 731 – 744, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0B-3VHWKSD-1/2/fb19de34342a419cdc098fcb1a890b27

[163] R. Pibernik and E. Sucky, "An approach to inter-domain master planning in supply chains," *International Journal of Production Economics*, vol. 108, no. 1-2, pp. 200–212, July 2007. [Online]. Available: http://dx.doi.org/10.1016/j.ijpe.2006.12.010

[164] C. Pütz and E. J. Sinz, "Model-driven derivation of BPMN workflow schemata from SOM business process models," *Enterprise Modelling and Information Systems Architectures*, vol. 5, no. 2, pp. 57–72, 2010.

[165] M. Reichert and P. Dadam, "Enabling adaptive process-aware information systems with ADEPT2," in *Handbook of Research on Business Process Modeling*, J. Cardoso and W. M. van der Aalst, Eds. Hershey, New York: Information Science Reference, March 2009, pp. 173–203. [Online]. Available: http://dbis.eprints.uni-ulm.de/476/

[166] H. A. Reijers and W. M. van der Aalst, "The effectiveness of workflow management systems: Predictions and lessons learned," *International Journal of Information Management*, vol. 25, no. 5, pp. 458 – 472, 2005.

[167] D. Robertson, A. Barker, P. Besana, A. Bundy, Y. Chen-Burger, D. Dupplaw, F. Giunchiglia, F. van Harmelen, F. Hassan, S. Kotoulas, D. Lambert, G. Li, J. McGinnis, F. McNeill, N. Osman, A. de Pinninck, R. Siebes, C. Sierra, and C. Walton, *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*, ser. Lecture Notes in Computer Science. Springer,

2009, vol. 4891, ch. Models of Interaction as a Grounding for Peer to Peer Knowledge Sharing, pp. 81–129.

[168] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "An MDA approach to develop secure business processes through a UML 2.0 extension," *International Journal of Computer Systems Science and Engineering, Special Issue: TrustBus 2006*, vol. 22, no. 5, pp. 307–319, Sep 2007.

[169] F. Rosenberg, C. Enzi, A. Michlmayr, C. Platzer, and S. Dustdar, "Integrating quality of service aspects in top-down business process development using WS-CDL and WS-BPEL," in *EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference.* Washington, DC, USA: IEEE Computer Society, 2007, p. 15.

[170] *RosettaNet Implementation Framework: Core Specification*, V02.00.01 ed., RosettaNet, www.rosettanet.org, March 2002. [Online]. Available: http://www.rosettanet.org/dnn_rose/DMX/tabid/2979/DMXModule/624/Command/Core_ViewDetails/Default.aspx?EntryId=4730

[171] RosettaNet, *Multiple Messaging Services (MMS) Profile for Web Services (WS) V11.00.01*, RosettaNet, August 2009.

[172] ——, *Message Control and Choreography (MCC) - Profile-Web Services (WS), Release 11.00.00A*, RosettaNet, June 2010.

[173] ——, *RosettaNet Methodology for Creating Choreographies, R11.00.00A*, RosettaNet, July 2011.

[174] N. Russell, W. van der Aalst, and N. Mulyar, "Workflow control-flow patterns: A revised view, BPM center report BPM-06-22," BPMcenter.org, Tech. Rep., 2006.

[175] T. Santoso, S. Ahmed, M. Goetschalckx, and A. Shapiro, "A stochastic programming approach for supply chain network design under uncertainty," *European Journal of Operational Research*, vol. 167, no. 1, pp. 96–115, November 2005. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v167y2005i1p96-115.html

[176] G. Scheithauer and G. Wirtz, "Case study: Applying business process management systems," in *Proceeding of the 2008 International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, Redwood City, CA (USA), July 2008.

[177] A. Schönberger, "Modelling and Validating Business Collaborations: A Case Study on RosettaNet," Otto-Friedrich-Universität Bamberg, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik 65, Mar. 2006. [Online]. Available: http://www.opus-bayern.de/uni-bamberg/volltexte/2006/81/pdf/modelFIN.pdf

[178] ——, “The CHORCH B2Bi approach: Performing ebBP choreographies as distributed BPEL orchestrations,” in *Proceedings of the 6th World Congress on Services 2010 (SERVICES 2010), Second International Workshop on Services Computing for B2B (SC4B2B), Miami, Florida, USA*. IEEE, July 2010.

[179] ——, “Do we need a refined choreography notion?” in *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS), Karlsruhe, Germany, February 21-22, 2011*, ser. CEUR Workshop Proceedings. CEUR-WS.org, Feb 2011, pp. 16–23.

[180] ——, “Visualizing B2Bi choreographies,” in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA’11), Irvine, California, USA*. IEEE, December 12-14 2011.

[181] A. Schönberger, T. Benker, S. Fritzemeier, M. Geiger, S. Harrer, T. Kessner, J. Schwalb, and G. Wirtz, “QoS-enabled business-to-business integration using ebBP to WS-BPEL translations,” in *Proceedings of the IEEE SCC 2009 International Conference on Services Computing, Bangalore, India*. IEEE, September 2009.

[182] A. Schönberger, C. Pflügler, and G. Wirtz, “Translating shared state based ebXML BPSS models to WS-BPEL,” *International Journal of Business Intelligence and Data Mining - Special Issue: 11th International Conference on Information Integration and Web-Based Applications and Services in December 2009*, vol. 5, no. 4, pp. 398 – 442, 2010.

[183] A. Schönberger, J. Schwalb, and G. Wirtz, “Has WS-I’s work resulted in WS-* interoperability?” in *Proceedings of the 9th 2011 International Conference on Web Services (ICWS 2011), Washington, D.C., USA*. IEEE, July 2011, pp. 171–178.

[184] A. Schönberger, C. Wilms, and G. Wirtz, “A Requirements Analysis of Business-To-Business Integration,” University of Bamberg, Technical Report: Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik 83, 12 2009.

[185] A. Schönberger and G. Wirtz, “Realising RosettaNet PIP Compositions as Web Service Orchestrations - A Case Study,” in *The 2006 International Conference on e-Learning, e-Business, Enterprise Information Systems, e-Government, & Outsourcing (CSREA EEE’06), Las Vegas, Nevada, USA*, June 26-29 2006, pp. 141–147.

[186] ——, “Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions,” in *The 2006 International Conference on Software Engineering Research and Practice (SERP’06), Las Vegas, Nevada, USA*, June 26-29 2006, pp. 329–335.

[187] ——, "Taxonomy on consistency requirements in the business process integration context," in *Proceedings of 2008 Conf. on Software Engineering and Knowledge Engineering (SEKE'2008).* Redwood City, California, USA: Knowledge Systems Institute, 1.-3. July 2008.

[188] ——, "Using variable communication technologies for realizing business collaborations," in *Proceedings of the 5th 2009 World Congress on Services (SERVICES 2009 PART II), International Workshop on Services Computing for B2B (SC4B2B), Bangalore, India.* IEEE, September 2009.

[189] ——, "Sequential composition of multi-party choreographies," in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA'10), Perth, Australia.* IEEE, December 13-15 2010.

[190] ——, "Towards executing ebBP-Reg B2Bi choreographies," in *Proceedings of the 12th IEEE Conference on Commerce and Enterprise Computing (CEC'10), Shanghai, China.* IEEE, November 10-12 2010.

[191] ——, "Configurable analysis of sequential multi-party choreographies," *forthcoming in International Journal of Computer Systems Science and Engineering*, March 2012.

[192] A. Schönberger, G. Wirtz, C. Huemer, and M. Zapletal, "A composable, QoS-aware and web services-based execution model for ebXML BPSS BusinessTransactions," in *Proceedings of the 6th 2010 World Congress on Services (SERVICES2010), Fourth International Workshop on Web Services and Cloud Services Testing (WS-CS-Testing 2010), Miami, Florida, USA.* IEEE, July 2010, pp. 229 – 236.

[193] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst, "Process flexibility: A survey of contemporary approaches," in *Advances in Enterprise Engineering I, 4th International Workshop CIAO! and 4th International Workshop EOMAS, held at CAiSE 2008, Proceedings*, ser. Lecture Notes in Business Information Processing, J. L. G. Dietz, A. Albani, and J. Barjis, Eds., vol. 10. Montpellier, France: Springer, June 2008, pp. 16–30.

[194] C. Schroth, T. Janner, and V. Hoyer, "Strategies for cross-organizational service composition," in *MCETECH '08: Proceedings of the 2008 International MCETECH Conference on e-Technologies.* Washington, DC, USA: IEEE Computer Society, 2008, pp. 93–103.

[195] K. Schulz and M. Orlowska, "Architectural issues for cross-organisational B2B interactions," in *International Conference on on Distributed Computing Systems Workshop 2001, Los Alamitos, California, USA*, apr 2001, pp. 79 –87.

[196] J. Schwalb and A. Schönberger, "Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations," Otto-Friedrich-Universität Bamberg, Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik 87, 09 2010.

[197] J. Schwalb, A. Schönberger, and G. Wirtz, "Approaching interoperability testing of QoS based on WS-* standards implementations," in *The 4th Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'10), co-located with 8th IEEE European Conference on Web Services (ECOWS 2010), Ayia Napa, Cyprus.* IEEE, December 2010.

[198] R. Seguel, R. Eshuis, and P. Grefen, "Constructing minimal protocol adaptors for service composition," in *WEWST '09: Proceedings of the 4th Workshop on Emerging Web Services Technology.* New York, NY, USA: ACM, 2009, pp. 29–38.

[199] S. Simmons, "Introducing the websphere integration reference architecture," *IBM WebSphere Developer Technical Journal*, vol. 8.5, pp. 5–20, August 2005. [Online]. Available: http://www.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html

[200] H. A. Simon, *The sciences of the artificial.* The MIT Press, 1996.

[201] M. P. Singh, "Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language," in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, Taipei, Taiwan*, ser. AAMAS '11. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 491–498. [Online]. Available: http://dl.acm.org/citation.cfm?id=2031678.2031687

[202] ——, "LoST: Local state transfer - an architectural style for the distributed enactment of business protocols," in *Proceedings of the 9th 2011 International Conference on Web Services(ICWS 2011), Washington, D.C., USA.* IEEE, July 2011, pp. 57–64.

[203] T. F. Stafford, "Introduction to special section on e-services," *Commun. ACM*, vol. 46, no. 6, pp. 26–28, June 2003. [Online]. Available: http://doi.acm.org/10.1145/777313.777333

[204] P. Telang and M. Singh, "Abstracting and applying business modeling patterns from rosettanet," in *Proceedings of the 8th International Conference on Service-Oriented Computing - ICSOC 2010, San Francisco, CA, USA*, ser. Lecture Notes in Computer Science, P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds. Springer Berlin / Heidelberg, December 2010, vol. 6470, pp. 426–440.

[205] P. Telang and M. P. Singh, "Specifying and verifying cross-organizational business models: An agent-oriented approach," *Services Computing, IEEE Transactions on*, vol. 4, no. 4, p. in press, 2011.

[206] A. J. C. Trappey, P.-S. Ho, T.-H. Lu, and S.-Y. Shih, "Building B2B protocols in inter-enterprise process execution," in *Proceedings of The 5th Asia-Pacific Industrial Engineering and Management Systems Conference, APIEMS 2004, Gold Coast, Australia*, 2004, pp. 7.3.1 – 7.3.9.

[207] UN/CEFACT, "UN/CEFACT' s Modelling Methodology N090 Revision 10," November 2001. [Online]. Available: http://www.untmg.org/

[208] ——, *UN/CEFACT's Modeling Methodology (UMM): UMM Meta Model - Foundation Module Version 1.0*, 1st ed., UN/CEFACT, 10 2006.

[209] ——, *Core Components Technical Specification Version 3.0*, UN/CEFACT, 9 2009.

[210] ——, *UML Profile for UN/CEFACT's Modeling Methodology (UMM) - Foundation Module*, 2nd ed., UN/CEFACT, 4 2011.

[211] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[212] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf, "From public views to private views - correctness-by-design for services," in *Proceedings of the 4th International Workshop on Web Services and Formal Methods, WS-FM 2007, Brisbane, Australia*, ser. Lecture Notes in Computer Science, M. Dumas and R. Heckel, Eds. Springer Berlin / Heidelberg, September 2007, vol. 4937, pp. 139–153. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-79230-7_10

[213] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, "Business process management: A survey," in *Business Process Management*, ser. Lecture Notes in Computer Science, W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, Eds., vol. 2678. Springer, 2003, pp. 1–12.

[214] W. M. P. van der Aalst and M. Weske, "The P2P approach to interorganizational workflows," in *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering.* London, UK: Springer-Verlag, 2001, pp. 140–156.

[215] W. M. van der Aalst and K. B. Lassen, "Translating unstructured workflow processes to readable BPEL: Theory and implementation," *Information and Software Technology*, vol. 50, no. 3, pp. 131 – 159, 2008.

[216] G. van Seghbroeck, F. de Turck, B. Dhoedt, and P. Demeester, “Web service choreography conformance verification in M2M systems through the pix-model,” in *Proceedings of the IEEE International Conference on Pervasive Services 2007, Istanbul, Turkey*, july 2007, pp. 385 –390.

[217] J. Vanhatalo, H. Völzer, and J. Koehler, “The refined process structure tree,” *Data Knowl. Eng.*, vol. 68, no. 9, pp. 793–818, 2009.

[218] W. Vogels, “Web services are not distributed objects,” *IEEE Internet Computing*, vol. 7, no. 6, pp. 59–66, 2003.

[219] W3C, *Web Services Description Language (WSDL) 1.1*, W3C, March 2001. [Online]. Available: http://www.w3.org/TR/wsdl

[220] ——, *XML Encryption Syntax and Processing*, W3C, December 2002. [Online]. Available: http://www.w3.org/TR/xmlenc-core/

[221] ——, “Web services architecture,” February 2004. [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf

[222] ——, *XML Schema Part 0: Primer Second Edition*, W3C, October 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-0/

[223] ——, *Web Services Choreography Description Language*, 1st ed., W3C, November 2005. [Online]. Available: http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/

[224] ——, *Web Services Addressing 1.0 - Core*, W3C, May 2006. [Online]. Available: http://www.w3.org/TR/ws-addr-core

[225] ——, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, W3C, April 2007. [Online]. Available: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[226] ——, *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, W3C, June 2007. [Online]. Available: http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/

[227] ——, *Web Services Policy 1.5 - Framework*, W3C, September 2007. [Online]. Available: http://www.w3.org/TR/2007/REC-ws-policy-20070904/

[228] ——, *XML Signature Syntax and Processing (Second Edition)*, W3C, June 2008. [Online]. Available: http://www.w3.org/TR/xmldsig-core/

[229] Y. Wand and R. Weber, “Research commentary: Information systems and conceptual modeling–a research agenda,” *Info. Sys. Research*, vol. 13, no. 4, pp. 363–376, 2002.

[230] I. Weber, J. Haller, and J. A. Mülle, "Automated derivation of executable business processes from choreographies in virtual organizations," in *In: F. Lehner, H. Nösekabel, P. Kleinschmidt (eds.): Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006), Band 2, XML4BPM Track, GITO-Verlag Berlin*, Mar. 2006, pp. 313–327.

[231] T. Weitzel, S. Martin, and W. König, "Straight through processing auf XML-Basis im Wertpapiergeschäft," *Wirtschaftsinformatik*, vol. 45, no. 4, pp. 409–420, 2003.

[232] S. Wieczorek, A. Roth, A. Stefanescu, V. Kozyura, A. Charfi, F. M. Kraft, and I. Schieferdecker, "Viewpoints for modeling choreographies in service-oriented architectures," in *Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture 2009 (WICSA/ECSA), Cambridge. Los Alamitos/Calif.*, 2009.

[233] G. Wirtz, M. Weske, and H. Giese, "The OCoN approach to workflow modeling in object-oriented systems," *Information Systems Frontiers 3:3*, pp. 357–376, 2001. [Online]. Available: citeseer.ist.psu.edu/wirtz01ocon.html

[234] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell, "On the suitability of BPMN for business process modelling," in *Proc. of the International Conference on Business Process Management (BPM 2006)*, ser. Lecture Notes in Computer Science, S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, Eds., vol. 4102. Springer, 2006, pp. 161–176. [Online]. Available: http://dblp.uni-trier.de/db/conf/bpm/bpm2006.html#WohedADHR06

[235] A. Wombacher, "Decentralized consistency checking in cross-organizational workflows," in *IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, EEE 2006, San Francisco, CA, USA*. IEEE Computer Society, June 2006, pp. 39–46.

[236] A. Wombacher, B. Mahleko, and E. Neuhold, "IPSI-PF. a business process matchmaking engine based on annotated finite state automata," *Information Systems and E-Business Management*, vol. 3, no. 2, pp. 127–150, 2005, 10.1007/s10257-005-0053-y. [Online]. Available: http://dx.doi.org/10.1007/s10257-005-0053-y

[237] WS-I, *Basic Profile Version 1.2*, WS-I, November 2010. [Online]. Available: http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html

[238] ——, *Basic Profile Version 2.0*, WS-I, November 2010. [Online]. Available: http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html

[239] ——, *Basic Security Profile Version 1.1*, WS-I, January 2010. [Online]. Available: http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html

[240] ——, *Reliable Secure Profile Version 1.0*, WS-I, November 2010. [Online]. Available: http://www.ws-i.org/profiles/ReliableSecureProfile-1.0-2010-11-09.html

[241] D. M. Yellin and R. E. Strom, "Protocol specifications and component adaptors," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, pp. 292–333, 1997.

[242] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and managing web services: issues, solutions, and directions," *The VLDB Journal*, vol. 17, no. 3, pp. 537–572, 2008.

[243] C. Yushi, L. Wah, and D. Limbu, "Web services composition - an overview of standards." Synthesis Journal, Fifth issue, ITSC publication, (pp 137-150), 2004. [Online]. Available: http://www.itsc.org.sg/synthesis/2004/4_WS.pdf

[244] J. Zaha, M. Dumas, A. ter Hofstede, A. P. Barros, and G. Decker, "Bridging global and local models of service-oriented systems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 3, pp. 302 –318, may 2008.

[245] J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Let's Dance: A language for service behavior modeling," in *Proceedings of the 14th international conference on cooperative information systems (CoopIS'06)*, Montpellier, France, 10 2006, pp. 145–162.

[246] M. Zapletal, "A holistic methodology for model-driven B2B integration: From business values over business collaborations to deployment artifacts," in *Proceedings of the Tenth International Conference on Electronic Commerce (ICEC08), August 2008, Innsbruck, Austria.* CEUR-WS.org, August 2008.

[247] ——, "A UML-based methodology for model-driven B2B integration: From business values over business processes to deployment artifacts," Ph.D. dissertation, Vienna University of Technology, June 2009.

[248] M. Zapletal, T. Motal, and H. Werthner, "The business choreography language (BCL) - a domain-specific language for global choreographies," in *Proceedings of the 5th 2009 World Congress on Services (SERVICES 2009 PART II), International Workshop on Services Computing for B2B (SC4B2B), Bangalore, India.* IEEE, September 2009.

[249] M. Zapletal, R. Schuster, P. Liegl, C. Huemer, and B. Hofreiter, "Modeling interorganizational business processes," in *Handbook on Business Process Management 1*, ser. International Handbooks on Information Systems, J. vom Brocke and M. Rosemann, Eds. Springer Berlin Heidelberg, 2010, pp. 543–564. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00416-2_25

[250] W. Zhao, B. R. Bryant, F. Cao, R. Hauser, K. Bhattacharya, and T. Tao, "Transforming business process models in the presence of irreducibility and concurrency," *International Journal of Business Process Integration and Management*, vol. 2, no. 1, pp. 37–48, 2007.

UNIVERSITY OF BAMBERG PRESS

The establishment and implementation of cross-organizational business processes is an implication of today's market pressure for efficiency gains. In this context, Business-To-Business integration (B2Bi) focuses on the information integration aspects of business processes. A core task of B2Bi is specifying the message exchanges between integration partners as so-called choreographies.

Despite the economic importance of B2Bi, existing choreography languages fall short of fulfilling all relevant requirements of B2Bi scenarios. Dedicated B2Bi choreography standards allow for inconsistent outcomes of basic interactions and do not provide unambiguous semantics for advanced interaction models. In contrast to this, more formal or technical choreography languages may provide unambiguous modeling semantics, but do not offer B2Bi domain concepts or an adequate level of abstraction. Defining valid and complete B2Bi choreography models becomes a challenging task in the face of these shortcomings.

The CHORCH approach offers B2Bi choreographies that are B2Bi adequate, simple, unambiguous, and implementable. Integration partners start out with a high-level visual model of their interactions in BPMN that identifies the types and sequences of the business document exchanges to be implemented. An ebBP refinement then is used to fill in technical details such that an implementation based on Web Services and BPEL can be derived.

ISBN 978-3-86309-076-0
ISSN 1867-7401
32,00 Euro