BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 90

Betsy - A BPEL Engine Test System

Simon Harrer, Jörg Lenhard

July 2012

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG

Distributed Systems Group

Otto-Friedrich Universität Bamberg

Feldkirchenstr. 21, 96052 Bamberg, GERMANY

Prof. Dr. rer. nat. Guido Wirtz

http://www.uni-bamberg.de/pi/

Due to hardware developments, strong application needs and the overwhelming influence of the net in almost all areas, distributed systems have become one of the most important topics for nowadays software industry. Owing to their ever increasing importance for everyday business, distributed systems have high requirements with respect to dependability, robustness and performance. Unfortunately, distribution adds its share to the problems of developing complex software systems. Heterogeneity in both, hardware and software, permanent changes, concurrency, distribution of components and the need for inter-operability between different systems complicate matters. Moreover, new technical aspects like resource management, load balancing and guaranteeing consistent operation in the presence of partial failures and deadlocks put an additional burden onto the developer.
The long-term common goal of our research efforts is the development, implementation and evaluation of methods helpful for the realization of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding distributed systems on all levels. Our current research activities are focussed on different aspects centered around that theme:

- *Reliable and inter-operable Service-oriented Architectures*: Development of design methods, languages, tools and middle-ware to ease the development of SOAs with an emphasis on provable correct systems that allow for early design-evaluation due to rigorous development methods. Additionally, we work on approaches and standards to provide truly inter-operable platforms for SOAs.
- *Implementation of Business Processes and Business-to-Business-Integration (B2Bi)*: Starting from requirements for successful B2Bi development processes, languages and systems, we investigate the practicability and inter-operability of different approaches and platforms for the design and implementation of business processes with a focus on combining processes from different business partners.
- *Quality-of-Service (QoS) Aspects for SOA and B2Bi*: QoS aspects, especially reliability and security, are indispensable when putting distributed systems into practical use. We work on methods that allow for a seamless observance of QoS aspects during the entire development process from high-level business processes down to implementation platforms.
- *Agent and Multi-Agent (MAS) Technology:* Development of new approaches to use Multi-Agent-Systems for designing, organizing and optimizing complex systems ranging from service management and SOA to electronic markets and virtual enterprises.
- *Visual Programming- and Design-Languages*: The goal of this long-term effort is the utilization of visual metaphors and languages as well as visualization techniques to make design- and programming languages more understandable and, hence, more easy-to-use.

More information about our work, i.e., projects, papers and software, is available at our homepage (see above). If you have any questions or suggestions regarding this report or our work in general, don't hesitate to contact me at `guido.wirtz@uni-bamberg.de`

Guido Wirtz

Bamberg, January 2010

BPEL
Engine
Test
SYstem

Betsy - A BPEL Engine Test System

Simon Harrer, Jörg Lenhard

Lehrstuhl für Praktische Informatik, Fakultät WIAI

`https://github.com/uniba-dsg/betsy`

Abstract More than five years have passed since the final release of the long-desired OASIS standard of a process language for web service orchestration, the *Web Services Business Process Execution Language (BPEL)*. The aim of this standard is to establish a universally accepted orchestration language that forms a core part of current service-oriented architectures and, because of standardisation, avoids vendor lock-in. High expectations, in academia and practice alike, have been set on it. By now, several fully conformant and highly scalable engines should have arrived in the market. The perception of many however, is that standard conformance in current engines is far from given. It is our aim to shed light on this situation. In this study, we present the tool *betsy*, a BPEL Engine Test System that allows for a fully-automatic assessment of the standard conformance of a given BPEL engine. We use it to examine the five most important open source BPEL engines available today. Betsy comes with a large set of engine-independent conformance test cases for assessing BPEL standard conformance. This enables us to give a view of the state of the art in BPEL support.

Keywords Service-oriented Architecture, BPEL, Engine, Open Source, Conformance Testing

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

BPEL	Web Services Business Process Execution Language
CLI	Command Line Interface
DSL	Domain Specific Language
ESB	Enterprise Service Bus
EPR	Endpoint Reference
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
SOA	Service-Oriented Architecture
SOAP	formerly known as Simple Object Access Protocol
URI	Universal Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Services Description Language
war	web archive
WSA	WS-Adressing
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSD	XML Schema Definition
XPath	XML Path Language

1 Purpose and Context

The BPEL specification [28] defines a language that can be used to build stateful Web Services that take part in long-running interactions which are typical in today's enterprise domain. As its name suggests, it is part of the larger Web Services architecture and relies strongly on other Web Services specifications, primarily the *Web Services Description Language (WSDL)* [40] in revision 1.1 and *SOAP* 1.1 [39], but it is also designed to be compatible with WS-*[1] specifications. Its main purpose being the construction of composite services from basic ones, BPEL is seen as a core part of Web Services-based Service-Oriented Architectures (SOAs) [30].

The advent of BPEL has been embraced by several scientific communities world-wide and many approaches for solving different problems in the area of distributed systems base on the standard. Such approaches implicitly assume that fully conformant engines are available one day. One typical subfield where BPEL is of relevance is the field of choreography modeling and execution that goes hand in hand with orchestration languages[2]. Roughly speaking, a choreography specifies a communication protocol between different distributed parties from a global point of view. Approaches for executing these communication protocols mostly involve a translation of the global protocol into several local models (top-down) or the construction of the global protocol from preexisting local models (bottom-up). For the implementation of the latter, BPEL is a natural candidate. Some approaches in this area, without claiming to be complete, are [7,8,16,18,29,32,45]. Another area where BPEL is closely scrutinized is that of process language suitability and expressiveness. Here, existing languages are assessed for their suitability for a certain domain based on their expressiveness, which is largely determined by the constructs available in the languages. Examples of studies in this area are [1, 2, 9, 10, 23, 36]. Finally, adapter synthesis is a field that concerns the generation of services to fix incompatibilities between stateful services whose communication protocols do not match [35]. Again, existing tool-chains [24, 25] assume that the different services are built in BPEL.

All these approaches benefit from the fact that BPEL is an open standard. It is free for anyone to use and, given multiple functioning and conformant engines implementing it are in place, it is a powerful tool to avoid vendor lock-in. If this premise is not fulfilled, the previous approaches release their full potential only in theory. Still, their practical usage would be desirable.

Here, we investigate the market of open source BPEL engines[3] and determine the degree of support for the BPEL specification that is in place today. To reach this aim, we have implemented the tool *betsy*. Betsy comprises a conformance testing suite that allows for the automatic assessment of the standard conformance of BPEL engines. This includes a large corpus of conformance tests that define standard-conformant behaviour, and a software system that can instrument these tests and execute them automatically for a given engine. Apart from providing correct results in a performant and readily understandable manner, the aim of this tool is to be easily extensible to include (i) an increased set of test cases, to obtain even more

[1]WS-* comprises a set of different standards for orthogonal aspects of service-oriented architectures. Notable standards are WS-Addressing [44] or WS-Security [27]. For more information on the Web Services architecture, please refer to [41].

[2][31] gives an introductionary overview of this area.

[3]We focus on open source engines, because they are generally more important to scientists. Nonetheless, the analysis of proprietary engines is important to practicioners and planned as future work.

significant results, and (ii) further engines, to get a more comprehensive picture of the current situation.

The testing of BPEL processes has so far attracted some interest in the research community[4]. Current approaches concentrate on three areas:

1. Unit testing of executable BPEL processes
2. Verifying the conformance of BPEL processes to a certain specification or formalism
3. Performance testing of BPEL processes or engines

The next paragraphs briefly discuss each of these areas and outline the similiarities and differences betsy shares with prominent approaches in each area. Basically, betsy builds on unit testing approaches and shares the aspect of testbed generation with performance testing approaches. It differs from conformance testing approaches in the system under test, being the middleware in the case of betsy and concrete process models in the case of other approaches. To the best of our knowledge, no other tool with the focus of the conformance testing of BPEL middleware can be found yet.

The area of unit testing BPEL processes is certainly the one that received the most interest, but even here more work is called for [46]. In this area, the BPELUnit[5] project [22] received the most wide-spread acceptance. BPELUnit is a member of the XUnit family, a set of unit testing frameworks where each framework targets the language X [15]. It allows for the construction of unit and integration tests for BPEL processes that run on specific engines. The aim of BPELUnit, and related unit testing approaches for BPEL, is to test and verify the correctness of specific BPEL processes. Betsy on the other hand, is aimed at the testing of the conformance of BPEL engines; that is, the systems under test are different. For the unit testing approaches it is a business application and for betsy it is the middleware infrastructure. Betsy is similar to BPELUnit in the manner that it allows for the automatic deployment, execution and verification of BPEL processes for specific engines. In fact, betsy internally uses a unit testing framework, *soapUI*[6], to automate the test execution and reporting, and builds its conformance testing workflow on top of that. We decided to use this framework instead of BPELUnit, because it has reached a more mature state and the set of engines we aimed to observe is intrinsically different to the set supported by BPELUnit. Nevertheless, betsy could be adapted to build on BPELUnit in the future which would offer the benefit of a higher richness of detail in the test configuration.

Conformance testing or conformance verification of BPEL is generally not understood as the testing of the conformance of a BPEL engine to the BPEL specification. Instead, the terms refer to the verification of the behavioural properties of a concrete BPEL process. For instance, it is verified that a concrete BPEL process behaves as specified by some abstract process model. Fields, such as business process modeling or choreography modeling (cf. above), have developed an extensive set of formalisms for the modeling of processes to which implementations,

[4]A comprehensive overview of academic approaches to web service testing is given in [5]. A subset of these approaches applies to BPEL and is discussed in the following.

[5]More details can be found at the project homepage: `http://bpelunit.github.com/introduction/`.

[6]SoapUI is a unit testing framework not specifically attached to BPEL, but to Web Services in general. It will be discussed more closely in the following sections.

often written in BPEL, have to conform. Alongside comes a large set of conformance notions that define different levels of conformance among these models[7]. Here, we do not focus on approaches for verifying behavioural conformance of concrete process models, but instead on implementation conformance of the middleware to the standard specification in the sense of [26, pp. 203-208],[20]. That is, a conformant implementation of the BPEL standard is an implementation that satisfies both, static and dynamic, conformance requirements. Such requirements are defined in the specification using the notational conventions [28, pp. 9/10] which follow the guidelines from [19].

The third area is performance testing of BPEL engines. To the best of our knowledge, this area has received only little attention and not many practical evaluations and case studies can be found. Nevertheless, perfomance is a leading factor when deciding about the usage of BPEL in mission-critical applications. The main approaches in this area are *SOABench*[8] [3] and *GENESIS2*[9] [21]. Both, SOABench and GENESIS2 are testbed environments that can be used to generate testbeds for complex service-oriented systems. Whereas GENESIS2 is directed at service-oriented systems in general, SOABench is specifically directed at the testing and analysis of the performance and scalability characteristics of BPEL engines. Consequently, in terms of the underlying domain models, betsy and SOABench have a larger intersection than betsy and GENESIS2. Both, SOABench and betsy treat the BPEL engines under test as black boxes and describe the test setup in an engine-independent domain model. Each tool uses its domain model to automatically generate and execute test cases and provides a plugin mechanism to extend the execution environment with new engines. However, the domain model of SOABench is more complex than that of betsy and includes the modeling of clients, physical machines and exchangable atomic services. Although this domain model is more complex to build, it also provides a more fine-grained control of the testing environment. This is required to enable SOABench to gather and compute performance metrics. As betsy is directed at conformance testing, it has no such requirements. Finally, whereas SOABench comes with four BPEL process definitions that are aimed at testing the performance and scalability of an engine, betsy comes with a set of almost 140 processes that have the aim to assess the standard conformance of an engine. Moreover, betsy natively supports five engines instead of three.

This report serves as an architectural white paper to the structure and functioning of betsy. It provides extensive results for five BPEL engines that serve as a demonstration of betsy's capabilities. The next section examines the engines under test. After that, we describe the architecture of the testing tool and the test cases we use to assess standard conformance. Finally, we give a detailed view of the results and discuss specialities of and implications for each of the engines.

[7]A subset of these models, which is not intended to be exhaustive, is [4, 12, 13, 33, 34].

[8]The project homepage can be found at `http://code.google.com/p/soabench/`.

[9]The project homepage is located at `http://www.infosys.tuwien.ac.at/prototyp/Genesis/Genesis_index.html`. Unfortunately as of July 2012, the sources for GENESIS2 are not provided, so it is not possible to build upon this tool.

2 Open Source BPEL Engines under Test

In the following, we provide a short description of the structure, architecture and setting of each of the engines under test, being *Apache ODE*, *bpel-g*, the *OpenESB BPEL Service Engine*, *Orchestra* and *Petals ESB*. All these engines are freely available, offer support for BPEL 2.0, and are still under active development today. Some of the engines claim to support both, BPEL 2.0 and its predecessor BPEL 1.1. As BPEL 2.0 is intended to replace BPEL 1.1, we focus solemly on BPEL 2.0 here. Interestingly, all engines are developed in Java.

	Apache ODE	Bpel-g	OpenESB	Orchestra	Petals BPEL SE
Version	1.3.5	5.3-snapshot	2.2	4.9	1.1.0
License	Apache	GPL	CDDL	LGPL	LGPL 2.1
Release Date	Feburary 2011	April 2012	December 2009	January 2012	February 2012
Programming Language	Java	Java	Java	Java	Java
Deployable WAR	X	X	O	X	O
Download Size	35.3 MB	31.9 MB	195 MB	125 MB	136 MB
Container	Tomcat 7.0.26	Tomcat 7.0.26	Glassfish v2	Tomcat 7.0.26	Petals ESB 4.0
Deployment	FS, WS, WI	FS, WS	CLI, WI	CLI, WS	FS, WS
Deployment Descriptor Format	ODE	ODE or BPR	JBI	-	JBI
Amount of Deploy-ment Descriptors	1	1	3	0	3

Table 1: General Engine Properties

Table 1 outlines several general properties about the engines under test. Most engines are available as a web archive (war) and can be deployed on a servlet container, for example Apache Tomcat[10]. Processes can be deployed to the engines using different mechanisms, outlined by the *Deployment* row. The different *deployment* options bear the following meaning:

FS: **F**ile **S**ystem; The engine supports *hot deployment* which means that processes are deployable by copying a deployment archive to a specific directory in the file system of the server.

WS: **W**eb **S**ervice; Once started, the engine offers a web service at a predefined endpoint to which deployment archives can be sent for deployment.

WI: **W**eb **I**nterface; Once started, the engine offers a web interface that can be accessed through a web browser. This interface has the option to select and upload a deployment archive.

Command Line Interface (CLI): **C**ommand **L**ine **I**nterface; The engine comes with tooling that can be invoked via the command line. This tooling offers the option to deploy a deployment archive.

There are several different formats, supported by varying engines, for the deployment descriptors that are required in the deployment archives. The *Deployment Descriptor Format* row outlines the most common formats we observered:

[10]The project homepage of Tomcat can be found at `http://tomcat.apache.org/`.

ODE: The Apache ODE project has developed a custom deployment descriptor format[11] that subsequently also gained support by other engines.

BPR: This is a custom format that was used by ActiveBPEL and has gained wider acceptance as it is used in ActiveVOS as well as bpel-g.

JBI: JBI 1.0 is a format based on JSR-208[12] to connect services which are deployed on service engines with endpoints through bindings via binding components. In our cases, the BPEL processes are deployed onto a BPEL service engine and connected to SOAP endpoints via a SOAP binding component using HTTP. This specification also received wider support by different engines and enterprise service buses (ESBs).

The following paragraphs briefly describe the origin and nature of each of the engines. The details and sources can be found at the respective project homepages, to which links are provided.

Apache ODE

As of today, Apache ODE is the most well-known and most widely used Open source BPEL engine available. It is maintained by the Apache Foundation[13] and supported among others by Intalio and JBoss. The engine is implemented in Java and relies on Jacob[14], a concurrency framework based on the actor model [17]. The most recent stable release at the time of writing, and the one used in this work, is ODE 1.3.5.

Bpel-g

The bpel-g engine is a derivate of the former ActiveBPEL by Active Endpoints[15]. Whilst ActiveBPEL is no longer available, bpel-g is still under development and maintained as a Google Code project[16]. The engine comprises the functionality provided by ActiveBPEL, but is extended to support and integrate with software libraries, such as Spring[17] or Apache Camel[18]. This analysis uses the 5.3 snapshot of bpel-g, being the most recent version available at the time. This is a beta release, and the stable release building on it may include additional features.

OpenESB

The OpenESB is an open Enterprise Service Bus that includes a BPEL engine[19]. It is written in Java and preceding its acquisition by Oracle, it was maintained by Sun. Today, its development is driven by LogiCoy[20] and Pymma Consulting[21]. OpenESB is commonly collocated with the

---

[11] See `http://ode.apache.org/creating-a-process.html#UserGuide-DeploymentDescriptor` for a description of this format.

[12] For a documentation of the format, see `http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html`.

[13] The project page is available at `http://ode.apache.org/`.

[14] More on Jacob can be found at `http://ode.apache.org/jacob.html`

[15] The company homepage is located at `http://www.activeendpoints.com/`.

[16] The project can be found at `http://code.google.com/p/bpel-g/`.

[17] See `http://www.springsource.org/` for details.

[18] See `http://camel.apache.org/` for details.

[19] For simplicity, by OpenESB we refer to its BPEL engine in this report.

[20] The company homepage can be found at `http://www.logicoy.com/`.

[21] The company homepage is available at `http://www.pymma.com/`.

Glassfish application server to form a full enterprise integration solution. The project homepage is `http://openesb-dev.org/` and the version used here is 2.2.

Orchestra

Orchestra is an open source BPEL engine released under LGPL and available at `http://orchestra.ow2.org/`. It is written in Java and developed by the OW2 consortium[22] and Bull. We analyze Orchestra 4.9, being the most recent stable revision at the time of writing. Orchestra executes BPEL on a generic process virtual machine. As listed in Table 1, Orchestra does not require a separate deployment package for deploying to an engine. Instead, it is sufficient to provide the BPEL and WSDL files directly. Although not being required, it is still possible to use a packaged format [6, pp. 21/22].

Petals ESB

Petals ESB is an open source ESB that includes a BPEL service engine[23] and a SOAP binding component. It is developed by the OW2 consortium, just as Orchestra, and is available at `http://petals.ow2.org/`. Instead of reusing Orchestra as a BPEL engine, Petals ESB provides a separate engine, namely EasyBPEL[24]. In the tests, we use EasyBPEL 4.0. Just like the other engines, EasyBPEL is written in Java.

[22] The consortium homepage is `http://ow2.org/`.

[23] For simplicity, by Petals ESB we refer to its BPEL engine in this report.

[24] Its documentation can be found at `http://research.petalslink.org/display/easybpel/EasyBPEL+Overview`.

3 System Architecture

Figure 1 outlines the overall structure of the testing tool. At root level, it contains build scripts and seperates into the packages for the build tool and the program code. The build tool we use is *Gradle*[25]. Gradle essentially is a Domain Specific Language (DSL) based on the scripting language Groovy[26] that combines the build systems Ant and Maven[27] with an object model tailored to the build process. The program and testing code is contained in the package

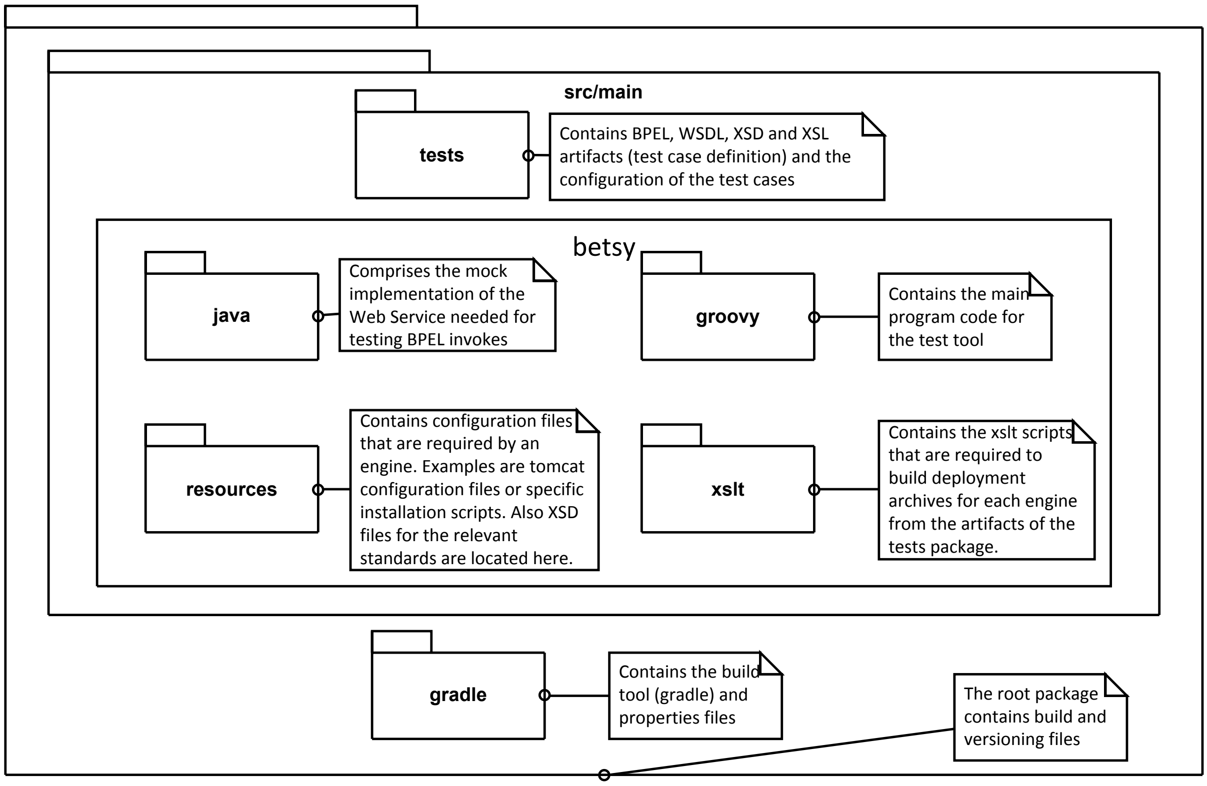

Figure 1: General Structure of Betsy

`src/main` and is itself split into the following packages:

tests The `tests` package comprises all test cases and their configuration. The test cases themself consist of BPEL, WSDL, XML Schema Definition (XSD), and eXtensible Stylesheet Language (XSL) files. Basically, the directory contains the resources betsy uses for generating the deployment archives for the engines under test, as well as for generating the corresponding soapUI test cases. The detailed description of the contents of this package is the purpose of section 5.

[25]For more detailed information, see `http://www.gradle.org/`.
[26]For more detailed information, see `http://groovy.codehaus.org/`
[27]Both, Ant and Maven are Apache projects and their documentation is available at `http://ant.apache.org/` and `http://maven.apache.org/` respectively.

groovy The `groovy` package contains most of the program code and forms the main body of betsy. It is dependent on an installation of *Ant* and *soapUI* on the local system. SoapUI is a mature unit testing tool for Web Services that is licensed under LGPL. It is available at `http://www.soapui.org/`. We use it so send SOAP messages to the service endpoints, to collect and analyze the replies, and to represent the outcome in a structured way. The `groovy` package implements the test execution process, being the process for installing the engines, generating the deployment archives and soapUI tests as well as executing the tests. A detailed description of the functioning of this software is given in section 4.

java The `java` package is required for the implementation of basic mock Web Services that are needed to test BPEL `invoke` activities. The classes and service interface in this package are generated using the `wsimport`[28] tool. The implementation of the interfaces and a publishing service is located in `groovy/betsy/executables/ws`.

resources The resources package contains additional files that are required by a specific engine. This can be a custom `log4j.properties` configuration for more detailed logging, as is the case for bpel-g and ODE. Other examples are custom installation scripts that are required for OpenESB. Finally, the package contains the XSD files for all relevant standards, such as BPEL, WSDL, and WS-Addressing.

xslt Building the deployment archives involves the construction of deployment descriptors for most engines. These descriptors are XML files that can be derived directly from the BPEL and WSDL files for a given test case. A convenient way to construct these XML files are XSL transformations. Such transformations are located within the `xslt` package, separated by engine. Additionally, due to the specifics of each engine, the WSDL or BPEL files may need to be modified. As an example, WSDL files are not allowed to contain `operation` elements which are not used in the corresponding BPEL process for bpel-g. For this purpose, XSL transformations are used, too.

[28]The `wsimport` tool is part of the JDK and can be found within its `bin` folder.

4 Structure and Execution of Betsy

The next section describes the architecture of the testing tool, followed by the structure of the test cases and their runtime representation. In section 4.3, the workflow for executing a test run is explained. In combination, these sections explain how to extend the tool with a new engine. Finally, section 4.4 describes how to download, install and run betsy.

4.1 Program Architecture

As stated in the previous section, the `groovy` package contains the main program code. This section clarifies the static structure and architecture of this package and provides hints on how to extend the software for the testing of further engines or further test cases. Its core parts, the data structures used to represent test cases and engines, and the classes that implement the runtime behaviour of the program, are detailed here.

The `groovy` package is structured as depicted in Figure 2. The core of this package, which

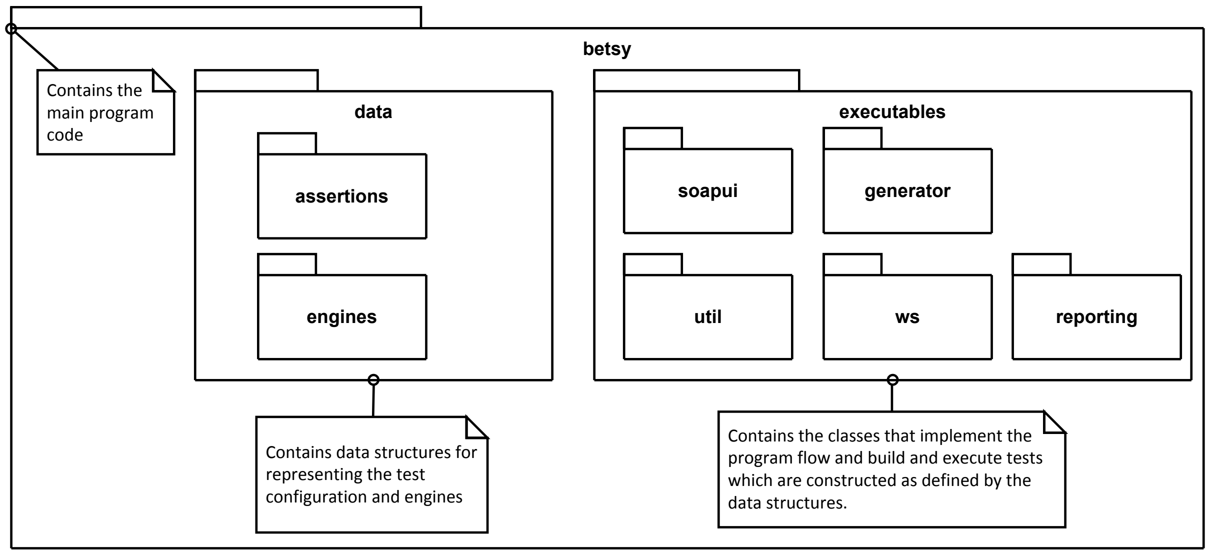

Figure 2: Structure of the `groovy` Package

is also the most relevant place when it comes to extending the software for another engine, comprises the `data` and `executables` packages.

The classes of the `data` package are used to represent test cases, test assertions and engines under test. Using a configuration of the instances of these classes, the `executables` can execute a corresponding test run and produce test reports. The sub-packages of the `data` package, `assertions` and `engines` include the classes for representing test assertions and engines respectively.

The classes of the `executables` package implement the workflow that is executed at runtime and described in section 4.3. The packages `soapui`, `util` and `ws` contained in `executables` provide classes with auxilliary functions. `Soapui` provides a wrapper that translates a test configuration, as defined by the data structures in `data`, into a soapUI project configuration. The package `util` consists of several helper classes, for example for measuring the execution time of the steps of a test run. The classes in `ws` implement and publish mock Web Services that are needed to test a BPEL `invoke` activity. The package `reporting` contains classes used to build report files, such as `csv` files, from the outcome of a test run. Finally, the classes for generating deployment archives can be found in the `generation` package.

4.2 Data Structures for Test Case Configuration

Figure 3 depicts the data structures that can be used to define a test configuration that can be executed by betsy.

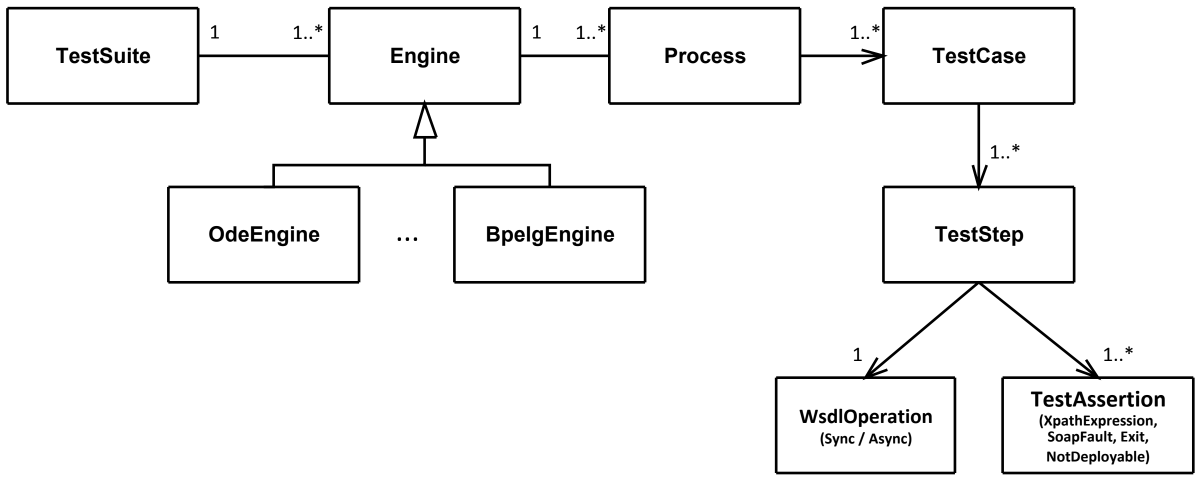

Figure 3: Data Structures for Defining a Test Configuration.

The starting point is a `TestSuite`. This is a top-level container that bundles multiple engines which are to be tested in a single run of betsy. The `Engine` class is abstract and defines the behaviour that must be implemented by a concrete engine, such as ODE or bpel-g to participate in the testing workflow. There is one concrete class for each engine under test. If betsy is to support a new engine, the `Engine` class must be extended and a new implementation for that engine must be provided. The following behaviour must be implemented:

Listing 1: Methods of an Engine

```
// Lifecycle methods for the engine itself
void install()
void startup()
void shutdown()
```

```

// lifecycle methods to deploy deployment archives on the engine
void deploy(process)

// getters for identifying folder names and generating endpoint urls
String getName()
String getDeploymentPrefix()
String getDeploymentPostfix()
```

An engine must provide the behaviour to install itself, which in most cases means coyping a war to a designated place in the file system. The `startup` and `shutdown` methods normally map to the execution of specific scripts shipped with the engine. Given the engine runs on tomcat, these methods map to the execution of the tomcat startup and shutdown scripts.

Finally, an engine must be able to deploy a given process as well as provide getter methods for the engine name and a deployment prefix (e.g. the hostname, port and url part at which processes deployed on that engine can be reached) and postfix (e.g. name tokens that are to be used by convention. For instance, the names of WSDL services need to end in `Service`).

Optionally, an Engine can override methods to hook into the execution process.

Listing 2: Optional Engine Methods

```
// Lifecycle methods for the engine itself
void failIfRunning()

// lifecycle methods to create deployment archives
void buildDeploymentDescriptors(process)
void transform(process)
void buildAdditionalArchives(process)
```

Most engines cannot be started if they already run and an attempt to do can disturb a currently executing test run. The method `failIfRunning` is used to detect whether an engine is already running and abort the whole test run if this is the case. This method is only called in the preparation phase and ensures that the relevant engines for that test run are not active.

The `buildDeploymentDescriptors` and `transform` methods are used to construct deployable artifacts from a set of engine-independent BPEL, WSDL and XSD files. The deployment descriptor must adhere to the format supported by the engine[29]. In all cases, this is a specific XML DSL and its concrete structure can be directly derived from the engine-independent files. Some engines also require specific transformations of the engine-independent files to support deployment which has to be implemented in the `transform` method. The `buildAdditionalArchives` method is called after the basic package is created. This is needed by some engines to build nested deployment archives.

Next, an engine is connected to a set of `Processes`. Each process references the files that are required for that process, being at least a BPEL and a WSDL file, but possibly also additional XSD or XSL files. Moreover, a process references several `TestCases` which define isolated tests of that process.

[29]The supported deployment descriptor formats are listed in Table 1.

A test case consists of multiple SOAP messages sent to an endpoint for a concrete service, each of which is defined in a `TestStep`. Each test step has a single `WsdlOperation` which can either be synchronous or asynchronous and a set of `TestAssertions`. Assertions are used to evaluate the outcome of a test step and to determine whether an engine supports a given feature. Given one assertion fails for a test step, the complete test case is marked as failure. Following assertions, located in `groovy/betsy/data/assertions`, are available:

`XpathTestAssertion` This assertion can be used to test a synchronous invocation. It selects a certain element in the response message and compares its content to a predefined value. If they are unequal, the test case is marked as failure.

`SoapFaultTestAssertion` Also this assertion can be used to test a synchronous invocation. It tests whether the response is a SOAP fault and optionally checks whether the response contains a given string (case sensitive).

`NotDeployableAssertion` Some test cases describe processes that must be rejected according to the BPEL standard. Generally, such cases should be detected by static analysis of the BPEL processor of an engine and not be deployed. This assertion verifies that the engine does not provide a WSDL at the expected location.

`ExitAssertion` The `exit` activity as well as the `exitOnStandardFault` attribute on the `process` element can terminate an instance immediatly. The BPEL standard does not specify the reaction of an engine if there are request-response operations which are still open at this time. This assertion tests for multiple different acceptable reactions observed for the engines, such as no response at all, a response with a HTTP error code of 200 and no response content, or a response containing "processTerminated".

Additionally, a test step can define a timeout after it has been executed. This allows to define delays between test steps. Such delays proved to be necessary as some engine need the time to correctly register correlation sets. Without the timeout, some tests for the correlation mechanism would fail otherwise. Finally, a description can be set for a `TestStep`, to enable meaningful error reporting if the step fails.

4.3 Test Generation and Execution

The workflow for generating and executing the test cases consists of several steps that are executed as shown in Figure 4 on the next page. At the beginning of every test run, the directory containing the deployment artifacts and reports of a previous test run are deleted. Next comes the testing of each process for each engine. This requires several substeps and is executed strictly sequential. This limitation ensures that the the execution of each test is not influenced by the parallel execution of another test and thus remains reproducible.

The first phase of the test execution is the generation phase, during which the deployment artifacts and soapUI test cases are generated. Next, the engine under test is installed and started. As soon as the engine is running, the previously generated deployment artifacts are deployed and the generated soapUI test cases can be run. Thereafter, the engine is stopped.

The aggregation of the reports, based on the outcome of the tests, takes place after all processes have been tested for all engines.

All these phases are explained in detail in the following sections. The input to this sequence of phases is formed by a list of engines and a list of processes. These are concrete instances of the classes described in Figure 3. The two lists are wired up and the set of processes, along with their test configuration, are executed for each of the engines.

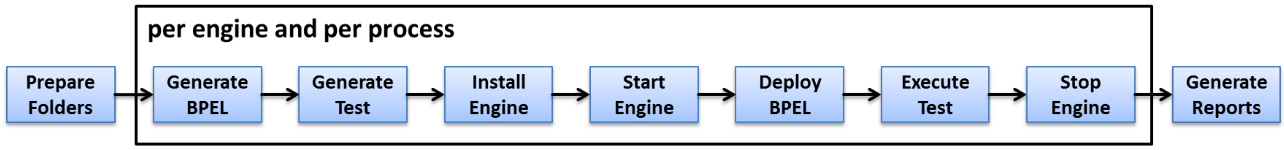

Figure 4: Test Execution Process

4.3.1 Preparation

The preparation phase is quite simple. It deletes the `test` folder of a previous run and recreates it afterwards. Additionally, it ensures that no engine that is to be tested is currently running. This is neccessary as engines cannot start if their ports are already in use.

4.3.2 Generation

The generation of the testing artifacts, being the deployable artifacts and the test projects, comprises two subsequent steps. The classes `PackageBuilder` and `TestBuilder` are responsible for the execution of these steps. The generation takes place for each process separately and leverages the run configuration of that process, as outlined in Figure 3 on page 10.

The deployment packages are created from the BPEL, WSDL and XSL files located in the `tests` directory. This generation involves the following steps:

Copy raw files: First, the raw files are copied into a specific directory for each engine and each test case in the test output directory. Per default, this is `/test/$Engine-Name/$Process-Name/bpel`.

Build deployment descriptor: Next the deployment descriptor is generated from the raw files. The generation of this descriptor is governed by each engine and implemented in XSL transformations. We use the ODE format for Apache ODE and bpel-g and the JBI format for openESB and Petals ESB. The only exception is Orchestra, which works without a deployment descriptor.

Apply transformations: Except for Orchestra, the combination of the raw files and the deployment descriptor is not yet deployable on an engine, because the service address in the raw file is a wildcard string and for execution it must be set according to an engine-specific naming convention. Bpel-g also requires the elimination of `operations` in the WSDL that are not matched by corresponding inbound message activities in the BPEL process. A deployment package with such a combination of files would be rejected upon deployment.

Package files: In all cases, the deployment package needs to be a single archive file. Most engines require a simply structured archive with the transformed files in the root folder of the archive. OpenESb and PetalsESB require a more complex structure with several additional archives included in the deployment package.

The second part of the generation phase is the generation of the soapUI test projects. One soapUI project is generated for each process. This project includes the execution of all test cases as defined by the test configuration. `TestSteps` translate to soapUI `WsdlTestRequests` and the `Assertions` are translated into XML Path Language (XPath) expressions, `ContainAssertions` or Groovy scripts for scanning the response of a `TestStep`, given it exists. This task is implemented in `betsy.executables.soapui.SoapUiWrapper` which encapsulates the soapUI API and automatically generates the test as defined by the given process.

4.3.3 Engine Installation and Startup

Each engine is installed separately by invoking an Ant task of the build file, `build.xml`. This file contains the following main targets.

```
Main targets:

 all        Install all BPEL engines
 bpelg      Install Bpel-g
 ode        Install Apache ODE
 openesb    Install OpenESB
 orchestra  Install Orchestra
 petalsesb  Install Petals ESB
Default target: all
```

The installation process of each engine works in the following steps:

Download required files: First, each engine is downloaded from the URL set in the build script. Currently, we provide the distributions used for the test results in this report at our department, but this may be subject to change. If another version of an engine is to be tested, it is sufficient to replace the download URL with another mirror. If the file is already present in the `downloads` folder, the download itself is skipped.

Delete previously installed engine: All engines are installed into the `server` folder located in the root folder of the tool. Given an engine has already been installed into this directory, it is deleted in this step. This is required to guarantee that a previous test run does not influence the current run. In a previous run, a certain process instance might have damaged the engine itself or its configuration. This can happen if an engine persists runtime data and there are bugs in the persistence mechanism.

Extract downloaded files: In this step, the downloaded files are copied in a newly-created and engine-specific folder in the `server` directory. Most engines come as archives, so they are extracted here. Moreover, Tomcat is extracted for engines that run on it and engine installation takes place by extracting the engine war file into Tomcat's `webapps` folder.

Customize installation: The engine installations are customized to collect more log messages from the engines under test. The files located in `src/main/resources/ENGINE_NAME` are copied to the corresponding destination for this reason.

For the details of the installation process of each engine, we refer to the Ant script.

Apache ODE, bpel-g, and Orchestra can be started by starting the Tomcat instance which the respective engines were installed in. Petals ESB and OpenESB can be started via their CLI.

As the BPEL engines under test tend to have high memory requirements, it is necessary to increase the amount of RAM allocated to the Java Virtual Machine (JVM) on which the engines run, especially for the engines using Tomcat. For the engines for which these requirements apply, an additional start script is provided which basically looks as follows:

```
SET "CATALINA_OPTS=-Xmx3048M -XX:MaxPermSize=2048m "
call normal tomcat start script
```

The first line sets the amount of memory used for this Tomcat JVM. The JVM can use up to 3 GB for elements that can be garbage-collected (e.g. instances of most classes), and up to 2 GB for elements that are never garbage-collected (e.g. classes or Strings). In the second line, the Tomcat startup script which ships with Tomcat itself is called.

4.3.4 Deployment and Execution

All engines deploy their artifacts sequentially. Apache ODE, bpel-g and Petals ESB allow for hot deployment. We use this deployment mechanism, as it is the fastest option. For Orchestra and OpenESB, the fastest deployment mechanism works via the CLI.

Once deployment is finished for a process on a specific engine, the execution phase starts. At first, two additional Web Services that conform to the WSDL definition used by the tests for the BPEL `invoke` activity are published using `TestPartnerServicePublisher`. These Web Services are implemented using `javax.jws` and run in the JVM of betsy itself. Next, the soapUI project corresponding to the current process is executed using the `SoapUiRunner` which acts as a wrapper for the soapUI class `SoapUITestCaseRunner`. SoapUI records all messages sent and received, as well as the outcome of the evaluation of the assertions in report files for each test case executed.

4.3.5 Shutdown

Once the test case has been executed, the corresponding engine is shut down. As each `Engine` has to implement the `shutdown` method, each engine can have a different shutdown strategy. For Apache ODE, bpel-g, and Orchestra, this is shutting down the Tomcat instance in which the respective engine runs. The fastest option is to use the command `taskkill` to kill the corresponding JVM. As each engine is installed again for each process, there is no problem in just killing the JVM. For PetalsESB this strategy applies as well, whereas OpenESB is shut down via an invocation of the CLI.

4.3.6 Reporting

After all test cases for all engines have been run, the reports are generated. The reports can be found in the `test/reports` directory. They are aggregated for the overall test run, but are also available in disaggregated form for each engine and process.

Report generation is implemented in the `betsy.executables.reporting.Reporter` class in four steps. First, the JUnit reports provided from the execution of the soapUI tests are enriched. The JUnit reports are merged with the text files containing detailed information about the messages exchanged. That way, it is possible to drill down from JUnit test reports to the corresponding message exchanges within the HTML reports generated in the next step. These reports are generated using an ant script leveraging the `junitreport` task. Due to the test package structure, the resulting report allows to drill down per engine and per test group. Thereafter, the test results are stored in `tests/reports/results.csv` for further automatic analysis. This file can be analyzed with any spreadsheet application. Lastly, LaTeX tables with the results and test case descriptions are generated. These tables can be found in sections A and B in the appendix.

4.4 Download and Installation of the Software

Betsy is available as an open source project on Github and can be checked out using `git clone git@github.com:uniba-dsg/betsy`. Following requirements must be met to execute betsy:

Working Internet Connection: The engines and required files are downloaded on demand during run-time. The following ports are used by the default configuration:

2000: Additional mock Web Services
8080: Orchestra
8080: Bpel-g
8080: Apache ODE
8084: Petals ESB
Various port in the range 4848 to 18181: OpenEsb

The fact that different engines can use the same ports is unproblematic. During a test run, at most one engine is running to avoid side-effects, so there are no port conflicts among the engines.

Windows 7: Betsy was developed in the Windows 7 operating system and has been verified to run in it. The tool is linked to the Windows operating system family, because engine startup and related tasks is implemented using `batch` scripts. Rewriting these scripts for another operating system should enable betsy to run on that system given the other requirements are met. A 64 bit system with at least 8 GB RAM is recommended.

JDK 1.7.0_03 (64 bit) or higher: `JAVA_HOME` must point to the JDK directory and `PATH` must include `JAVA_HOME/bin`.

Ant 1.8.3: `ANT_HOME` must point to the Ant directory and `PATH` must include `ANT_HOME/bin`.

SoapUI 4.5.0 (64 bit): SoapUI 4.5.0 must be installed to `C:\Program Files\SmartBear\soapUI-4.5.0`. The soapUI libraries need to be available at compile time.

Given these requirements are met, the tool can be started by passing a set of engines and processes to its core class, `src/main/groovy/betsy.Betsy`. Main classes for this task are available in `src/main/tests/configuration`. Another option to automatically run all test cases on all engines is to execute the command `gradlew run` in the root folder of the downloaded project. Gradle automatically downloads all necessary dependencies (including itself) and no further configuration is required.

5 Test Cases

The test cases are an essential part of the software and consist of two main components. The first one comprises the test case definitions, being raw standard-conformant BPEL, WSDL, XSD, or XSL files. The second one is made up of the test case configuration, the definition of message exchange sequences as well as inputs and expected outputs, and is implemented in Groovy classes. The structure of the test case definitions is explained in section 5.1. The test case configuration is discussed in the section thereafter.

5.1 Test Case Definition

The test case definitions are made up of BPEL, WSDL, XSD and XSL files that are required to derive the deployment artifacts for a specific engine. These test files themselves are derived from the relevant specifications [28, 37–40, 42–44] and are constructed conforming strictly to them, in an engine-independent manner. Each test case definition describes a specific feature of the BPEL specification, in a relatively isolated fashion, including files from other standards when needed.

5.1.1 Structuring and Scope

All engine-independent test files are located in the `src/main/tests` package. Figure 5 on the facing page depicts the contents of this folder.

basic-activities This package contains the test cases for all activities that are declared as basic activities in the BPEL specification; that is, those activities described in section 10 of the specification [28, pp. 84–97]: `invoke`, `receive` and `reply`, `assign`, `throw`, `wait`, `empty`, `exit`, `validate` and `rethrow`. The cases cover different configurations of these activities in relative isolation and test for good cases as well as edge conditions and faults, verifying the correctness of the behaviour inspite of faults. Tests for the following BPEL standard faults[30] are included: `xsltInvalidSource`, `xsltStylesheetNotFound`, `subLanguageExecutionFault`, `invalidExpressionValue`, `invalidVariables`, `unsupportedReference`, `selectionFailure`, `correlationViolation`, `conflictingReceive`, `ambiguousReceive`, `conflictingRequest` and `uninitializedVariable`. Finally, the package contains tests for the use of `extensionActivities`.

structured-activities This package contains the test cases for all structured activites defined in section 11 of the specification [28, pp. 98–114]. These are `sequence`, `if`, `while`, `repeatUntil`, `pick`, `flow` and `forEach`. Just like in the specification, although being structured, `scopes` are treated seperately. Again, good cases, specific edge conditions and problematic configurations are tested for, as well as faults typical for structured activities. Such faults are `joinFailure`, `invalidExpressionValue`, `invalidBranchCondition`, and `completionConditionFailure`.

[30]For a comprehensive list of BPEL standard faults, see [28, 192/193].

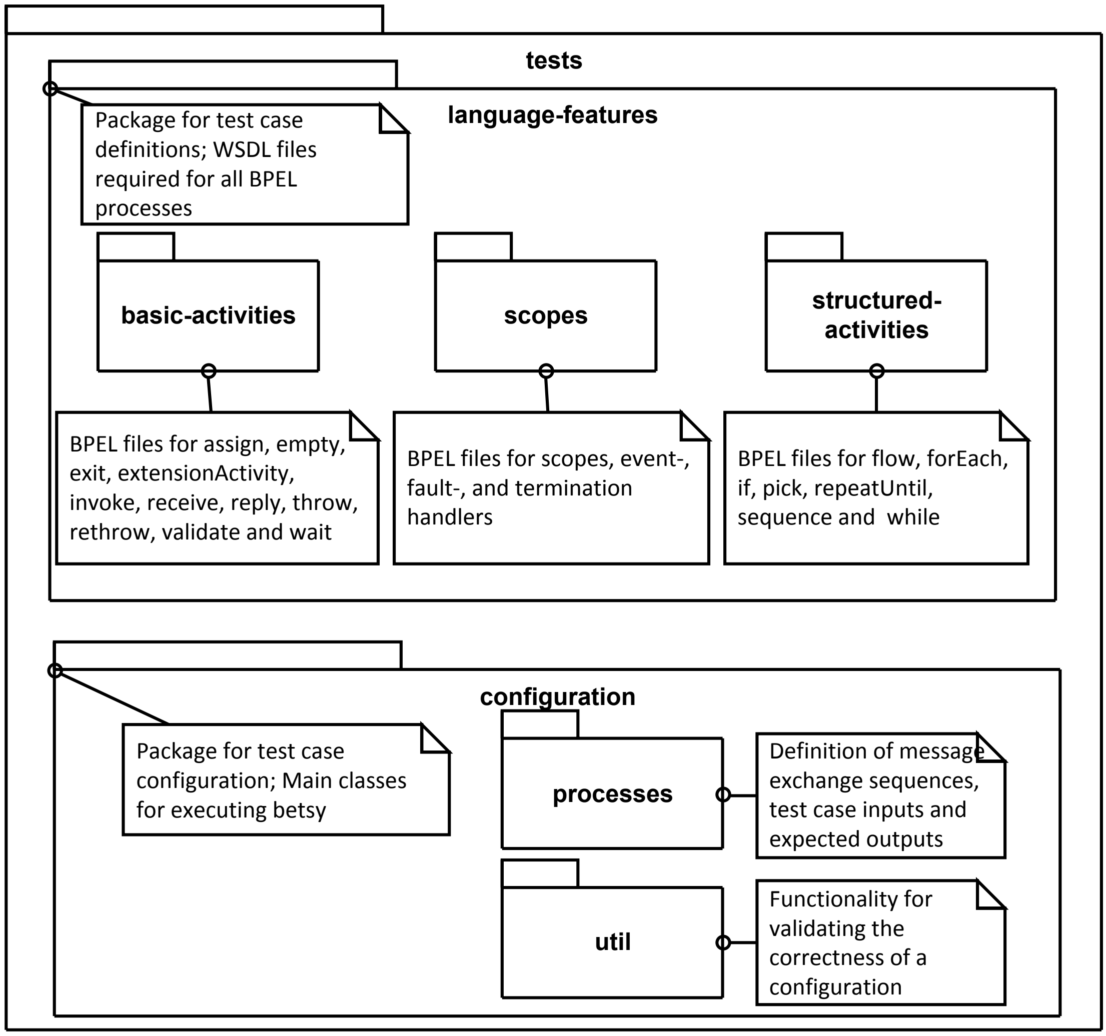

Figure 5: Structure of the Tests Package

scopes The `scopes` package contains all tests for attributes specific for `scope` activities and related constructs, such as `fault-`, `compensation-`, `termination-`, and `eventHandlers`, as defined in [28, pp. 115-147]. Practically all these contstructs can be used at process-level `scope` and do not require the explicit use of the `scope` construct. The tests for `faultHandlers` comprise tests for different configurations of the `catch` and the `catchAll` constructs. The `compensationHandler` tests also include the activities `compensate` and `compensateScope`, which are strictly speaking basic activities that can only be applied in `faultHandlers`. The `eventHandler` tests examine the two types of `eventHandlers`, `onMessage` and `onAlarm`. Also this package tests for several standard faults, in particular `joinFailure`, `missingReply`, and `missingRequest`. Finally, the package includes tests for the scope-level definition and overwriting of several constructs that are typically defined at process-level. These are `variables`, `partnerLinks`, `correlationSets` and `messageExchanges`.

There are no additional test packages for the aspects of sections 6 to 9 of the specification[31] [28, pp. 36-83]. As hinted in the preceding description, these aspects are tested for in the context of activity tests. They are generally not (conformance-) testable in complete isolation and a working process can hardly be built without leveraging several of these features. `PartnerLinks` are a necessity for communicating with a process instance in the first place and `variables` and data handling are needed to make sense of the process behaviour and verify its conformance to the specification. `CorrelationSets` are not required in all cases, just in those that involve multiple, especially asynchronous, message exchanges.

The tests of BPEL standard faults [28, pp. 192/193] are also distributed over the test packages and included according to the best fit. The faults `uninitializedPartnerRole` and `scopeInitializationFailure` are not tested for in this setup. The problem here is that the provocation of these faults is basically engine-specific. For example, there is no standard way to construct a process definition that forces an engine to throw a `scopeInitializationFailure` during process execution. A variety of errors that could provoke this fault are typically detected during static analysis and prohibit deployment, or produce other faults, such as `invalidVariables`. Although it is possible to forbid the initialization of a `partnerLink` on process instatiation, an engine is free to initialize it during its first use, thus completely avoiding the fault `uninitializedPartnerRole`.

It is important to emphasize once more that our tests are conformance tests; that is, the tests try to verify that constructs and combinations thereof that are defined by the BPEL specification [28], are supported and behave as specified. We denote this as *positive support* which is our main focus here. Except for some important edge cases, we do not test that certain process definitions that must be rejected according to the specification are also rejected. This behaviour describes *negative support*. Positive support of a vast part of a specification is the prerequiste for a useable piece of software and negative support is not equally critical in this context. The current state of the engines under test[32] suggests that positive support is still far from sufficient, rendering tests for correct negative support less important. Nevertheless, future work will improve this situation. As we test for standard conformance, we do also exclude any concrete extensions an engine may provide. A certain feature, such as the execution of XSL transformations or the assignment of `partnerLinks` may be supported by an engine, but in a fashion that does not conform to the specification. Although this increases the functionality of an engine, it renders the process definitions for this engine hardly portable and at least partly defeates the usage of open standards in the first place.

5.1.2 Test Interface and Example

To make the test case definitions instrumentable for betsy and to streamline their execution, all process definitions follow a common schema. First, every test case shares the same interface; that is, each process reuses the same WSDL definition as `myRole` `partnerLink`. This definition can be found in listing 3.

[31] These sections cover `partnerLinks`, `variables`, data handling and message correlation.

[32] See section 6 for the details.

Listing 3: Test Case WSDL Interface

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TestInterface"
             targetNamespace="http://dsg.wiai.uniba.de/bpel-engine-comparison/activities/wsdl/testinterface"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:plink="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
             xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
             xmlns:tns="http://dsg.wiai.uniba.de/bpel-engine-comparison/activities/wsdl/testinterface">

    <plink:partnerLinkType name="TestInterfacePartnerLinkType">
        <plink:role name="testInterfaceRole" portType="tns:TestInterfacePortType"/>
    </plink:partnerLinkType>

    <vprop:property name="correlationId" type="xsd:int"/>
    <vprop:propertyAlias messageType="tns:executeProcessSyncRequest" part="inputPart" propertyName="tns:correlationId"/>
    <vprop:propertyAlias messageType="tns:executeProcessSyncResponse" part="outputPart" propertyName="tns:correlationId" />
    <vprop:propertyAlias messageType="tns:executeProcessAsyncRequest" part="inputPart" propertyName="tns:correlationId"/>

    <types>
        <xsd:schema targetNamespace="http://dsg.wiai.uniba.de/bpel-engine-comparison/activities/wsdl/testinterface" xmlns:tns="http://dsg.wiai.uniba.de/bpel-engine-comparison/activities/wsdl/testinterface">
            <xsd:element name="testElementSyncRequest" type="xsd:int"/>
            <xsd:element name="testElementAsyncRequest" type="xsd:int"/>
            <xsd:element name="testElementSyncResponse" type="xsd:int"/>
            <xsd:element name="testElementSyncFault" type="xsd:int"/>
        </xsd:schema>
    </types>

    <message name="executeProcessSyncRequest">
        <part name="inputPart" element="tns:testElementSyncRequest"/>
    </message>
    <message name="executeProcessAsyncRequest">
        <part name="inputPart" element="tns:testElementAsyncRequest"/>
    </message>
    <message name="executeProcessSyncResponse">
        <part name="outputPart" element="tns:testElementSyncResponse"/>
    </message>
    <message name="executeProcessSyncFault">
        <part name="payload" element="tns:testElementSyncFault"/>
    </message>

```

```
    <portType name="TestInterfacePortType">
        <operation name="startProcessAsync">
            <input name="asyncInput" message="tns:executeProcessAsyncRequest"
                />
        </operation>
        <operation name="startProcessSync">
            <input name="syncInput" message="tns:executeProcessSyncRequest"/>
            <output name="syncOutput" message="tns:executeProcessSyncResponse
                "/>
            <fault name="syncFault" message="tns:executeProcessSyncFault"/>
        </operation>
    </portType>

    <binding name="TestInterfacePortTypeBinding" type="
        tns:TestInterfacePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.
            org/soap/http"/>
        <operation name="startProcessAsync">
            <soap:operation soapAction="async"/>
            <input name="asyncInput">
                <soap:body use="literal"/>
            </input>
        </operation>
        <operation name="startProcessSync">
            <soap:operation soapAction="sync"/>
            <input name="syncInput">
                <soap:body use="literal"/>
            </input>
            <output name="syncOutput">
                <soap:body use="literal"/>
            </output>
            <fault name="syncFault">
                <soap:fault name="syncFault" use="literal"/>
            </fault>
        </operation>
    </binding>

    <service name="TestInterfaceService">
        <port name="TestInterfacePort" binding="
            tns:TestInterfacePortTypeBinding">
            <soap:address location="ENDPOINT_URL"/>
        </port>
    </service>

</definitions>
```

The structure of this definition is intended to be as minimalistic as possible, while preserving the ability to adequately test all features of BPEL. The `partnerLinkType` definition is required to use the WSDL definition as a BPEL `partnerLink`. `Properties` and `propertyAliases` need to be in place to test correlation and asynchronous messaging. Next, the `types` and corresponding `message` definitions are very simple. Basically, all messages consist of a single message part that is of type `xs:int` [43]. By using a message part this simple, we intend to avoid problems resulting from the use of non-conformant or inadequate XML processors for data handling by an engine. Conformance tests of the XML processing capabilities of the engines, or rather their XML processors, are important, but not our focus here and should be conducted separately[33]. The `portType` we use for invoking the processes contains two operations, an asynchronous and a synchronous one that may return a fault. This is all that is needed for testing any sequence of message exchanges which can be constructed based on BPEL's correlation mechanism. In most test cases[34], the correctness of a test is assessed by interpreting the result of a synchonrous invocation. The `binding` we use is of `document|literal` style. This type of binding is relatively simplistic and thus has a high potential of being supported by an engine. Using this binding, SOAP messages are send as is, in a single document, without any encoding or additional wrapper elements, as it would be the case with other styles such as `rpc|encoded` [40, Sec.3]. In essence, this is the most basic SOAP binding currently available.

Listing 4 demonstrates a BPEL process that is used to test the support for the `sequence` activity. Obviously, the test consists of more structures than said activity, so the test is not strictly isolated. Yet, this is normal for a conformance test [26, pp. 203-208].

Listing 4: Test Case for the `sequence` Activity

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="Sequence"
    targetNamespace="http://dsg.wiai.uniba.de/bpel-engine-
        comparison/activities/bpel/sequence"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:ti="http://dsg.wiai.uniba.de/bpel-engine-
        comparison/activities/wsdl/testinterface">
    <import namespace="http://dsg.wiai.uniba.de/bpel-engine-
        comparison/activities/wsdl/testinterface" location="../TestInterface.
        wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>

    <partnerLinks>
        <partnerLink name="MyRoleLink" partnerLinkType="
            ti:TestInterfacePartnerLinkType" myRole="testInterfaceRole"/>
    </partnerLinks>

    <variables>
     <variable name="ReplyData" messageType="ti:executeProcessSyncResponse"/>
     <variable name="InitData" messageType="ti:executeProcessSyncRequest"/>
    </variables>

    <sequence>
        <receive name="InitialReceive" createInstance="yes"
```

[33] Although XML processors are relatively stable today, this is a real issue for the engines at hand. Even the simple data types used here revealed problems in the XML processing capabilities. Most notably, Apache ODE did not correctly process `xs:int`, converting it to `xs:decimal` instead and also replying decimal values where integer values are required by the schema definition. For the details, see section 6.1

[34] This naturally excludes the test cases that specifically test asynchronous communication.

```
        partnerLink="MyRoleLink" operation="startProcessSync"
        portType="ti:TestInterfacePortType" variable="InitData"/>
        <assign name="AssignReplyData">
            <copy>
                <from variable="InitData" part="inputPart"/>
                <to variable="ReplyData" part="outputPart"/>
            </copy>
        </assign>
        <reply name="ReplyToInitialReceive" partnerLink="MyRoleLink"
           operation="startProcessSync" portType="ti:TestInterfacePortType"
           variable="ReplyData"/>
    </sequence>

</process>
```

Every BPEL process must have a `partnerLink` defined as `myRole` and must import the according `partnerLinkType` and WSDL definition for this `partnerLink`. Here, this is the WSDL definition described before. Moreover, every process definition needs to have a start activity that performs the process instance creation. In BPEL, this is always an inbound message activity. The most basic inbound message activity is the `receive` activity. Used in a basic fashion, it has the highest chance of being correctly supported and, therefore, is unlikely to interfere with the actual feature under test. This assumption can be verified with designated `receive` activity tests and has been proven correct for the engines at hand[35]. The process uses a synchronous operation to have output available that can be interpreted to verify the correctness of the test. For this reason, variables for in- and outbound messages and an `assign` and a `reply` activity, for assigning the output data and for sending the response, are required. It would be possible to use an asynchronous operation with no inbound variables and no `reply` and, strictly speaking, such a test case would test a language feature in a more isolated fashion. However, such a test case would not return a result that can be evaluated to verify the correctness of the implementation. Instead, it would be necessary to evaluate engine logs or similar resources to make that decision. This would strongly link the testing procedure to the concrete engines and make the tool hard to extend. As demonstrably the simple synchronous invocation works for all engines under test, we are safe to use it as the basis for the conformance tests without corrupting the results.

5.2 Test Case Configuration

The artifacts from the previous section are sufficient to automatically build engine-specific deployment packages, but they are not yet sufficient for conformance tests. What is missing is the configuration of the test execution; that is the definition of inputs, message exchange sequences, and desired outputs. This configuration can be built using the classes described in section 4.2. The classes that build the configuration are located in `tests/configuration`, in parallel to the test case definitions. Most importantly, a `Process` comprises all necessary configuration for a single test. This structure is depicted in Figure 5 on page 19. In the

[35] For the details, see section 6.

`processes` package, several classes that construct valid test case configurations can be found. In its root, the `configuration` package contains main classes that use these configuration classes to construct a set of complete conformance tests and thereafter hand over that configuration to betsy to execute it. Listing 5 exemplifies two test case configurations.

Listing 5: Test Case Configuration Examples

```
1 public final Process SEQUENCE = builder.buildStructuredActivityProcess(
2   "Sequence", "A receive-reply pair enclosed in a sequence.",
3     [
4       new TestCase(testSteps: [new TestStep(input: "5", output: "5",
            operation: WsdlOperation.SYNC)])
5     ]
6 )
7
8 public final Process FLOW_LINKS_JOIN_CONDITION = builder.
      buildStructuredActivityProcess(
9   "Flow-Links-JoinCondition", "A receive-reply pair with an intermediate flow
        that contains three assigns, two of which point to the third using
       links. Both links have transitionConditions and their target a
       joinCondition defined upon them. A joinFailure should result, given not
        both of the links are activated.",
10      [ new TestCase(testSteps: [new TestStep(input: "1", assertions: [new
            SoapFaultTestAssertion(faultString: "joinFailure")], operation:
            WsdlOperation.SYNC)]),
11        new TestCase(testSteps: [new TestStep(input: "3", output: "6",
              operation: WsdlOperation.SYNC)])
12      ]
13 )
```

As demonstrated in the listing, `processes` are constructed using *factory methods* [11]. There is one such method that provides the necessary configuration for every test case definition. Lines 1 to 6 show the cofiguration of the sequence test case. In line 1, a builder is used to construct the configuration for a structured activity. Line 2 first sets the name of the BPEL file, here *Sequence*, relative to the test case definition directory of the structured activities. The builder ensures that all necessary files, such as WSDL files, are referenced in the process configuration. The second part of line 2 sets a description of the process. Next come the `TestCases` that are to be executed. For the sequence test, there is a single `TestCase`, consisting of a single `TestStep`. The `TestStep` requires the execution of the synchronous operation and the usage of an input of 5 in the message sent to the service. It expects the response message to include the value 5 as content for the proper message part. Only if that is true, the test case is successful. Otherwise, it is marked as failed.

Lines 8 to 13 show a more complex configuration for testing the support for `links` in the `flow` activity, including `joinConditions`. Apart from a more lengthy description of the process, there are two `TestCases`. Both consist of a single `TestStep` that uses the synchronous operation. However, they differ in the input they provide and the output they expect. The first `TestStep` uses an input of 1, which is a special number that should make the process instance throw a fault, and expects the response to be SOAP that contains the string *joinFailure*. The second `TestStep` uses an input of 3 and expects the response to be a regular SOAP message with a content of 6.

5.3 Restrictions

Certain specific aspects required in executable BPEL processes, namely partner reference schemes, the usage of XSL style sheet resources and fault propagation in the case a process instance faults with open inbound message activities are not defined in the specification. Strictly speaking, this means that features like the assignment of `partnerLinks` or the usage of XSL stylesheets cannot be implemented in a standard-conformant and portable manner. We decided to test these features nonetheless, focusing on reasonable reference and addressing schemes.

When it comes to the assignment of `partnerLinks`, the specification requires this as a mandatory feature, but only defines a container for a service reference, `service-ref` [28, pp. 38/39]. The concrete format of the reference is left open, the only restriction is that an engine should throw an `unsupportedReference` fault, if it does not understand the format. A reasonable and portable format to use here is a WS-Addressing `EndpointReference` [44] which is what we test for.

Furthermore, if a BPEL process uses a XSL transformation, the style sheet should be identified by a Universal Resource Identifier (URI) [28, p. 63]. The structure of this URI however, is left open. We decided to use the name of the style sheet as URI and to put the style sheet in the same place where the other deployment files go.

Finally, the BPEL specification does not require a certain behavior in case a process fails at top-level with open inbound message activites. The fault handling mechanism of virtually any high-level programming language like Java or C# would propagate the fault to the caller. Such behaviour is also a requirement for distributed fault handling in the first place [14]. This mechanism is adopted by all but one of the engines under test. The conformance tests check for the fault names to appear in the response of a request-response interaction, expecting that a fault is propagated to the caller.

6 Results

The following sections describe the result of an execution run of betsy for the complete test set and all engines described in section 2. This provides a detailed outline of the degree of support each of the engines provide for the BPEL specification. Naturally, this degree of support may change in future revisions of an engine. Using betsy, such changes can be easily detected.

The test run described here was configured as follows: All test cases were executed strictly sequential for all engines. There was no parallelism between the execution of the tests of the different engines. Before each test case for an engine, the engine was deleted and completely reinstalled from scratch. The intent of this configuration, lacking any parallelism at the cost of a high execution time, is to guarantee that no test case execution is influenced by a previous test case execution for the same engine or by parallel test case executions for another engine. In our setup, the execution time of this configuration was about twelve hours. To start this test run, execute the following command in the root folder of the project:

```
gradlew run
```

An overview of the support is given for each engine separately. The complete result tables can be found in the appendix in section A. The appendix also contains test descriptions and configurations for every test case which can be found in section B. Each test description follows the schema presented in Table 3.

<table>
<tr><td>Process name</td><td>The name of the test which outlines the feature under test.</td></tr>
<tr><td>Activities and configuration</td><td>A list of the BPEL activities, special constructs, and attributes used in the test.</td></tr>
<tr><td>Description</td><td>A short description of the structure and functioning of the test.</td></tr>
<tr><td>Test case: One row for every case associated with the test. Every test case consists of at least one test step</td><td><table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>The input value used in this test step</td><td>The operation used in this test step. This can be synchronous or asynchronous</td><td>The assertions used to verify the correctness of the result. This can for example be an expected output value (output: 5) or a fault (fault: xsltInvalidSource).</td></tr>
<tr><td>optionally: wait for</td><td>Time in milliseconds</td><td></td></tr>
</table></td></tr>
<tr><td>Support: The result of the test per engine. - marks a failure of all test cases, +/- a partial failure, and + a success of all test cases.</td><td><table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>-</td><td>+</td><td>-</td><td>-</td></tr>
</table></td></tr>
</table>

Table 3: Schema of a Test Description

Every test description starts with the name of the test, which provides a short outline of the feature under test. The test name is also identical to the file and process name that form the test definition. Next comes a list of all activities and special constructs or configurations that are used in the test definition. Certain elements that reoccur in every test definition, such as `process`, `import`, `partnerLinks`, `variables` and elements that are mandatory given certain other elements are used, such as `finalCounterValue` and `startCounterValue` in case the process uses a `forEach`, are excluded from this list. A full listing of the excluded elements can be found in section C in the appendix. Thereafter comes a short description of the test that outlines the structure of the test definition and special behavioural properties. Then follows a list of all test cases for the given test; that is, a list of all connected sequences of test steps. For each test step, the value used in the message sent to the process and the operation, being asynchronous or synchronous, and the assertions are listed. The assertions are of the types described in section 4.2. Examples are an expected output value for the proper message part or a SOAP fault that contains a certain string. Given a single assertion of any test step of a test case fails, the overall test case is marked as failure. A test step can define a wait period to delay the test execution for a specific amount of time. If this is the case, a row is added after the test step stating the timeout in milliseconds. In the end comes the support rating for all engines. The support rating is the metric that defines whether an engine successfully passed the test. The rating is trivalent and has the domain of -, if all test cases failed, +/- if at least one test case was successful and +, if all test cases were successful.

It needs to be noted that there may be false positives in the results. This is can occur for example when testing that a process must be rejected by an engine. Such a case is asserted by verifying that the WSDL file is not present at a given Uniform Resource Locator (URL). The problem is that we cannot assert *why* the WSDL file is not present. It can result from the BPEL process being rejected correctly or because the process uses yet another feature of BPEL which always leads to a deployment error for that engine. Therefore, the results have to be interpreted with care.

The next sections give an overview of the results for each of the engines. They discuss main findings and highlight areas of strong or weak conformance. This forms a good start for understanding the detailed results for each engine, contained in the appendix.

6.1 ODE Results

ODE itself describes its standard compliance on its homepage[36]. In the following, we focus solemly on our results. ODE is rated third in the overall BPEL support rating. Its major shortcoming is the unsupported initialization of `correlationSets` using asynchronous operations. For this reason, each test using `correlationSets` is available in two variants, one initializing them with an asynchronous operation and one with a synchronous operation. The name of the tests using asynchronous operations end with `-InitAsync` while the ones using synchronous operations end with `-InitSync`. This allows to test other features relying on `correlationSets` independently of this ODE specific error. The second most important reason for ODE failing a test is that it did not deploy a BPEL process. Due to its internal BPEL compiler[37], many test processes are rejected. This compiler has a list of BPEL constructs which are not supported by ODE and rejects any process definition which contains any of these con-

[36] For the details, see `http://ode.apache.org/ws-bpel-20-specification-compliance.html`

[37] The tool can also be called separately by invoking the `bin/bpelc.bat` script. For more information, see `http://ode.apache.org/bpelc-command.html`

structs. The third most important reason for a failing test is a timeout while waiting for the response to a corresponding request. However, we could not find a pattern hinting to a specific unimplemented or buggy feature or activity linked to this timeout. There may be multiple sources for such timeouts instead.

structured activities: The `flow` activity is supported in full, including `links`, `Join-` and `TransitionConditions` as well as the optional `suppressJoinFailure` attribute. However, ODE accepts process models that should be rejected according to the BPEL specification. This is the case for a BPEL process which defines start activities and non-start activities to execute in parallel. The `forEach` activity is only supported in sequence or in parallel without any `CompletionConditions` and without validations of invalid values for the `startCounter` and `stopCounter`. Interestingly, ODE catches variable duplication errors and rejects deployment. The `if` activity along with `else` and `elseif` are supported while invalid expression values within a condition of an `if` activity are not detected. Initializing `correlationSets` during an asynchronous `receive` leads to an internal `NullPointerException` as can be seen in the console log. An infinite loop occurs when the termination condition of a `RepeatUntil` activity is based on comparison with the equality operator.

basic activities: As mentioned in the previous paragraph, `correlationSets` cannot be initialized in conjunction with an asynchronous operation. Therefore, all tests ending with `-InitAsync` fail and typically allmost all tests ending with `-InitSync` succeed. However, there is one exception. When having initialized a `correlationSet` in a synchronous `receive` activity and using this `correlationSet` for an asynchronous receive, the same `NullPointerException` occurs.

ODE does not support `toParts` and `fromParts` for `invoke` or `receive` activities. Such processes are rejected during deployment as ODE itself is aware of that fact. This also applies for empty `from` or `to` elements in `copy` blocks as well as `doXslTransform` functions used in `assign` activities. Processes using any form of XSD validation, either a `validate` activity or a `validate` attribute within an `assign` are not rejected but the validation is simply ignored during execution. The same holds for the use of the `keepSrcElementName` attribute within an `assign` activity.

ODE does not throw correct faults (or any fault at all) in case of ambigious or conflicting receives or requests. The same is true for any `correlationViolation` fault. Throwing and rethrowing is implemented in a standard conformant way. However, their counterparts, namely the `catch` and `catchAll` activities cause timeouts during execution when attached to an `invoke` activity. When a request-response connection is open and an intermediate invoke receives a fault as response, this fault is not propagated to the outer scopes.

Variables cannot be initialized with a default value as the corresponding initialization block is ignored and the fault `uninitializedVariable` is thrown.

Despite its internal BPEL compiler, ODE does not reject processes referencing `ExtensionActivity` elements using `mustUnderstand='yes'` but ignores them instead. Thus, the inverse test using `mustUnderstand='no'` succeeds. The assignment of `partnerLinks` is not supported in conjunction with WS-Adressing (WSA). However, it does work in a more simplified manner, by copying the Endpoint Reference (EPR) url to the `partnerLink`.

scopes: ODE does not support `terminationHandlers` and ignores such handlers during execution. The faults `missingReply` or `missingRequest` are not thrown when they should be. When using the `parts` element within an `eventHandler`, the process is rejected during deployment. This is caused by the internal compilation with the `bpelc` tool. ODE basically ignores `exitOnStandardFault` attribute and if a fault occurs it is always thrown to the caller regardless of its type. The `compensate` activity in conjunction with the `compensationHandler` is works, except for the case of compensating a `scope` which has been instantiated multiple times within a `while` activity. In that case, only the first `scope` is compensated instead of all `scopes` that have been enabled during the execution of the `while` activity.

6.2 Bpel-g Results

In the overall support rating, bpel-g offers the highest degree of standard conformance; that is, it passes more test cases than the other engines. Especially the error handling of the engine is more standard-conformant than it is for other engines. Bpel-g more often throws the BPEL faults that must be thrown in a certain situation.

Apart from the SOAP faults that are expected in some test cases, bpel-g shows two typical errors that can happen during the execution of a process. These errors mark situations where the execution of a test case failed unexpectedly. The first one seems to apply if there is a problem with the handling of the input to a process at some stage during its execution. In this case, the process instance crashes without a standard BPEL fault. The fault code of this error is `soapenv:Server.userException` and the fault string is the quite generic `org.activebpel.rt.bpel.AeBusinessProcessException`. The other error also marks an unexpected crash of a process instance and results in a SOAP fault. Bpel-g's behaviour in this case is, in principle, not distinguishable from the previous error. However, the faultcode sent reads like a more problematic issue, indicating problems in the engine itself. This faultcode is `systemError`.

structured activities: The `flow` activity is supported in full, including `links`, `Join-` and `TransitionConditions` and the `suppressJoinFailure` attribute. However, bpel-g also allows for process models where start activities and non-start activities could be executed in parallel. Such a process definition should be rejected according to the specification. Bpel-g does not support time-related activities in a conformant manner. The `pick` activity with a `onAlarm` handler does not properly process the contents of neither a `for`, nor an `until` element. The tests indicate that both handlers do never fire when executed with an `xs:dateTime` or `xs:duration`. Bpel-g does not properly verify the counter variable of a `forEach` activity on its initialization. The counter can be initialized with values too large for `xs:unsignedInt`. Moreover, negative values are not processed entirely correct for this activity. Although such a case should throw an `invalidExpressionValue` fault, it results in an `invalidBranchCondition` fault during execution when evaluating a negative completionCondition. Finally, bpel-g allows for the deployment of process definitions with variable duplication errors. Such process definitions should be rejected by static analysis. Apart from these deficiencies, bpel-g passes all structured activity tests.

basic activities: There are several problematic spots in bpel-g's support for basic activities. Just like the other engines, bpel-g does not support empty `from` or `to` elements in `copy` blocks and instead fails with one of its custom exceptions. Furthermore, bpel-g does not read XSL style sheets identified as defined in section 5.3, but it does inform about this fact with the correct SOAP fault. Remarkably, bpel-g is the only engine that supports the assignment of a `partnerLink` with a WS-Addressing `endpointReference`, although it does not throw the expected `unsupportedReference` fault if a bogus reference scheme is used, but replies with a `systemError`. Bpel-g ignores extensions that it does not understand, even if the attribute `mustUnderstand` is set to true. In the latter case, deployment should fail instead. When it comes to the invocation of Web Services, bpel-g supports most invariants. That is synchronous and asynchronous invocation, `faultHandlers`, although faults that are thrown by the invoked web service are not handled correctly and can also not be caught by name in a `catch`. Furthermore, `compensationHandlers`, the use of `correlationSets`, `fromParts` and `toParts` are supported in an `invoke`. The only broader lack in Web Service invocation is that bpel-g is not able to invoke operations that expect empty messages. The `receive` activity and its counterpart `reply` also show high support. Here, the only problem is correct fault handling. `ConflictingReceiveFault` and `correlationViolation` are partly not thrown when they should have been or result in the typical `userException` instead. Finally, the timing activity `wait` and its `for` and `until` elements are again not supported using the standard XSD data types. All other tests for basic activities pass correctly.

scopes: Bpel-g's support for `scopes` and FCT-handlers is fairly comprehensive. The only bigger problem spots are as before time-related event handlers, which again are unable to deal with XSD data types, and `terminationHandlers` which are not supported at all. All other tests pass correctly.

6.3 OpenESB Results

The OpenESB BPEL service engine shows a fair degree of standard conformance and is the engine ranking second in the overall comparison. In case there are unexpected crashes of a process instance, the engine replies with a generic failure with the fault code `SOAP-ENV:Server`. The fault string carries additional information like stack traces or, in some cases, the BPEL fault.

structured activities: The OpenESB supports the `flow` activity in a relatively limited fashion. To be exact, `links` are not supported by this engine. The OpenESB accepts process definitions that include links, but ignores their usage. This can lead to the case that a process definition is executed correctly, but only if the activities in the `flow` are defined in the same order as the precedence relationsship specified by the `links`. Thus, this behaviour is coincidental. The `parallel` attribute for the `forEach` activity is ignored, just like `links`, and results in a sequential execution. Also, counter values exceeding `xs:unsignedInt` and duplicate variable definitions are not properly handled in this context. When it comes to conditions, the engine seems to evalute any value that is not a `xs:boolean`, say an arbitrary string, to true, instead of rejecting it with an `invalidExpressionValue`. Finally, the OpenESB has problems in handling the correlations of a `pick` activity. Given, the operation used in the `onMessage` of the `pick` is already used before in the same process instance, the operation fails with the `SOAP-ENV:Server` fault and the string of this fault indicates a `correlationViolation`. This happens, even if the correlation set used in said `onMessage` is initiated and used correctly.

basic activities: The usage of XSL stylesheets is not possible with OpenESB and processes using the respective XPath function are not deployed. The `keepSrcElementName` attribute of a `copy` element is ignored and the assignment of `partnerLinks` with `EndpointReferences` does not work. The documentation of OpenESB states that WSA `EndpointReferences` are supported, but links to the initial submission version of WSA and not the final recommendation. So, this may be a simple namespacing issue, where the OpenESB expects an `EndpointReference` with an outdated namespace. Nevertheless, the engine correctly throws an `unsupportedReference` fault for a bogus referencing scheme. Process definitions that include extensions which are not understood by OpenESB are ignored and deployed, even if `mustUnderstand` is set to `yes`. The `parts` syntax is not supported for messaging activities and the same applies to `fault-` and `compensationHandlers`, as well as the `correlations` elements for the `invoke` activity. The invocation of Web Service operations with an empty message is unsupported, too. If no fault data is used, faults thrown by `throw` or `rethrow` activities are propagated to the caller correctly. Given faults occur in messaging activities or the throwing activities use fault data, the OpenESB does not reply with the respective fault, but with its custom fault. Finally, the default initialization of variables is not supported, resulting is an `uninitializedVariables` fault, on the first usage of a variable.

scopes: OpenESB's support for scope-related aspects is the most comprehensive of all engines under test. It only fails to support `missingReply` and `missingRequest` faults, `fromParts` in `eventHandlers` and scope-level definition of `correlationSets`.

6.4 Orchestra Results

Orchestra takes a somewhat different approach to fault propagation than the other engines. That is, Orchestra does not propagate faults to the caller if a process instance fails with a fault at root level and still has open request-response conversations. Instead, faults are only propagated if specifically requested. If a fault is thrown and not handled in the context of the process instance, the engine replies with a HTTP 200 status code and zero length content. The fault only reaches a caller, if it is explicitly propagated using a `reply` activity. As discussed before, this mechanism contrasts current fault-handling practices and severly hampers distributed fault handling. It is also hardly possible to diagnose the origin of an error.

structured activities: Orchestra's support for the `flow` activity is quite comprehensive, including `links` with `joinConditions` and `transistionConditions`. The only aspect that could not be verified, due to Orchestra's fault handling strategy, is the handling of `joinFailures`. The `forEach` activity, however, is not supported at all. Orchestra forbid's the deployment of process definitions that include this activity, marking the test for the rejection of a definition with duplicate variables as a false positive. Finally, Orchestra seems unable to initiate a `correlationSet` with an asynchronous operation.

basic activities: Because of its fault propagation strategy, Orchestra fails a large set of the basic activity tests. The cases where a test makes a process instance terminate with a fault, and where an interal failure resulting from a software bug crashes the instance, are undistinguishable with this strategy. For this reason, Orchestra fails all tests for the `throw` and `rethrow` activities, as well as all other tests that are expected to result in a fault. A basic level of support could be verified for the `assign` activity, where the `ignoreMissingFromData` attribute, the `getVariableProperty` function, `from` and `to` elements, literal assignment, and queries are correctly supported. Orchestra rejects any process definition with an extension it does not understand. This is the correct behaviour, if the `mustUnderstand` attribute is set to `yes`, but wrong if it is left default. The `invoke` activity supports invocation with empty messages, the `fromParts` syntax, and, interestingly, also `faultHandlers`. This aspect indicates that fault handling is indeed in place and only the fault propagation mechanism is debatable. `CompensationHandlers` are not supported for the `invoke` activity, however. The `receive` and `reply` activities work in a basic fashion, including the `parts` syntax and `messageExchanges`, but the correctness of fault handling could not be verified. The `wait` activity and the default initialization of variables work, but the `validate` activity is unsupported.

scopes: The `faultHandler` test cases show that Orchestra internally handles faults, at least as long as no `faultVariables` are used. Basic support for `compensationHandlers` seems to be in place, although there are issues in invoking these handlers. The simple test cases fail with Orchestra's typical fault behaviour, but a more complex test case with multiple nested scopes passes. These results suggest that Orchestra has problems if a `compensationHandler` is attached to a scope directly under the root scope, but not if it is attached to a deeper nested scope. The scope-level definition of `correlationSets` is not supported. `OnMessage eventHandlers` lack support for the `parts` syntax and `onAlarm` handlers do not allow the `repeatEvery` attribute in conjunction with `for` or `until`.

6.5 Petals ESB Results

Petals ESB supports only a minimal set of activities and is ranking lowest in the overall ranking. More than 75% of the tests fail. The three most common errors observed in the tests are:

- A valid BPEL process could not be deployed successfully.
- A timeout occured during message exchange.
- A SOAP fault occured referencing a Petals ESB namespaced fault string with a java stack trace caused by a `javax.jbi.messaging.MessagingException`.

All three errors are very generic and do not give much information about the initial cause of the error. Thus, for some features, the reasons why they do not work cannot be detected. For instance, no test case using `correlationSets` was successful and the SOAP fault containing the `MessagingException` is thrown. It is unclear whether this is an internal bug or whether `correlationSets` simply have not been implemented. Regardless of the root of this error, lacking support for `correlationSets` leaves this engine unsuitable for asynchronous messaging.

structured activities: The support for structured activities in Petals ESB is limited to the very basic control flow constructs in their default configuration. The `if` activity with its `else` and `elseif` additions is supported in full while the `forEach` activity is functioning in parallel and in sequence only. The most basic activity, namely `sequence`, is supported as well as the `while` loop. A `pick` can only be used to create an instance without a correlation set.

basic activities: Petals ESB can `receive` and send (`invoke`) messages synchronously as well as asynchronously in a basic fashion. A `catchAll` block attached to an `invoke` is also supported, but all variants of message exchanges are not. Interestingly however, Petals ESB is the only engine which rethrows a fault from an invoke, given there are open request-response operations. The copying of data using the `assign` activity works when using expressions, properties or literals and the activity also supports the `ignoreMissingFromData` attribute. Variables can be initialized with default values and the `empty` and `exit` activities work as expected. The `wait` activity only works in conjunction with the `until` condition and the usage of `for` leads to a timeout.

scopes: Petals ESB's support for scopes is limited to `faultHandlers` which work in any combination except for the access of the fault variable in the `catch` element. Additionally, the `exitOnStandardFault` attribute is supported in part. A process definition with this attribute set to `yes` behaves correctly as long as no `joinFailure` is thrown.

7 Summary

Table 4 shows the aggregated number of successful, partially successful and failed test cases by engine, as well as the resulting ranking. Bpel-g achieves the highest amount of standard conformance, followed by OpenESB and ODE close up. Orchestra comes fourth and the fifth position is taken by Petals ESB, which shows a relatively limited degree of support.

	bpel-g	ODE	openESB	Orchestra	PetalsESB
successful tests	106	82	85	63	32
partially successful tests	0	0	2	1	0
failed tests	28	52	47	70	102
Rank	1	3	2	4	5

Table 4: Overall Test Results

It is striking that hardly any activity is supported by all engines. Looking at the basic activities, only basic configurations of `assign`, `empty`, `exit`, `invoke`, `receive`, and `reply` are supported by all engines. The same applies to `if`, `sequence`, and `while`, as well as `faultHandlers`. This implies that relatively basic communication and control-flow facilities are in place, but there is no common support for more than the absolute minimum. Porting BPEL processes from an engine to another one, therefore, is a daunting task.

Betsy eases this task considerably by revealing the capabilities of each of the engines and assessing their degree of standard conformance in a comparable and uniform manner. Still, there are plenty of areas where betsy and the assessment of the standard conformance of BPEL engines can be improved. For instance, the test set used by betsy, although being quite comprehensive, does not cover every single aspect defined in the BPEL specification. One area are the static analysis features described in appendix B of the specification [28, pp. 194–205]. These features describe process definitions that should be rejected by a BPEL engine and currently betsy tests only a few of them. Moreover, all the current test cases test single features. It should be interesting to include complex realistic use cases that use a variety of features in combination. It can be expected that features that work in relative isolation may stop working when used in combination with other constructs. Another important aspect is the testing of more engines, especially commercial ones, such as the Oracle Process Manager or IBM Websphere. Complementary standards and languages, revolving around BPEL, form a third area of future work. As indicated by the measurements here, the XML processing capabilities of at least one engine are problematic. Moreover, the support for standards such as WS-Security and WS-ReliableMessaging may be crucial in a business application and the WS-Stack that an engine builds upon should be evaluated for these capabilities as well.

All in all, betsy comes a long way in terms of the conformance evaluation of BPEL engines. We have demonstrated its applicability and its current state forms an excellent position for tackling further issues. We aim to extend the tool with more functionality and hope that it can be of benefit to the research community. The test results attached to this report and especially the test cases that ship with betsy have the potential to improve any of the engines under test. They can be used to track down bugs that hamper a certain feature and can also be used for regression testing. Using betsy, this testing process can be easily automated. Last but not least, the results provide an interesting snapshot of state of the implementation of BPEL.

References

[1] A. P. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE)*, pages 245–259, Braga, Portugal, March/April 2007.

[2] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Service Interaction Patterns. In *3rd International Conference on Business Process Management*, pages 302–318, Nancy, France, September 2005.

[3] D. Bianculli, W. Binder, and M. L. Drago. Automated performance assessment for service-oriented middleware: a case study on BPEL engines. In *Proceedings of the 19th international conference on World wide web*, pages 141–150, Raleigh, North Carolina, USA, April 2010.

[4] A. Both and W. Zimmermann. Automatic Protocol Conformance Checking of Recursive and Parallel BPEL Systems. In *European Conference on Web Services (ECOWS)*, pages 81–91, Dublin, Ireland, November 2008. IEEE.

[5] M. Bozkurt, M. Harman, and Y. Hassoun. Testing & Verification In Service-Oriented Architecture: A Survey. *Software Testing, Verificaton and Reliability*, 00:1–7, May 2012.

[6] Bull SAS – OW2 Consortium. *Orchestra User Guide*, October 2011.

[7] G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *Proceedings of the IEEE 2007 International Conference on Web Services (ICWS)*, pages 296–303, Salt Lake City, Utah, USA, July 2007.

[8] G. Decker, O. Kopp, F. Leymann, and M. Weske. Interacting services: From specification to execution. *Data & Knowledge Engineering, Elsevier*, 68(10):946–972, 2009.

[9] G. Decker and J. Mendling. Process Instantiation. *Data and Knowledge Engineering, Elsevier*, 68:777–792, 2009.

[10] G. Decker, H. Overdick, and J. Zaha. On the Suitability of WS-CDL for Choreography Modeling. In *Proceedings of Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen (EMISA)*, pages 21–33, Hamburg, Germany, October 2006.

[11] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Amsterdam, 1995. ISBN: 0201633612.

[12] J. García-Fanjul and J. T. Claudio de la Riva. Generation of Conformance Test Suites for Compositions of Web Services Using Model Checking. In *Testing: Academic and Industrial Conference – Practice And Research Techniques*, Windsor, United Kingdom, August 2006. IEEE.

[13] M. Geiger, A. Schönberger, and G. Wirtz. Towards automated conformance checking of ebBP-ST choreographies and corresponding WS-BPEL based orchestrations. In *Conference on Software Engineering and Knowledge Engineering (SEKE), Miami, Florida, USA*. Knowledge Systems Institute, 7.-9. July 2011.

[14] C. Guidi, I. Lanese, F. Montesi, and G. Zavattaro. On the Interplay Between Fault Handling and Request-Response Service Interactions. In *8th International Conference on Application of Concurrency to System Design*, pages 190–198, Xi'an, China, June 2008.

[15] P. Hamill. *Unit Test Frameworks*. O'Reilly, 2004. ISBN-13: 978-0596006891.

[16] 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.

[17] C. Hewitt, P. Bishop, and R. Steiger. A universal modular ACTOR formalism for artificial intelligence. In *International Joint Conference on Artificial intelligence*, IJCAI'73, pages 235–245, San Francisco, CA, USA, 1973.

[18] B. Hofreiter and C. Huemer. A model-driven top-down approach to inter-organizational systems: From global choreography models to executable BPEL. In *Join Conf CEC, EEE*, 2008.

[19] IETF. *Key words for use in RFCs to Indicate Requirement Levels*, March 1997. RFC 2119.

[20] ISO. *ISO/IEC 9646-1:1994 – Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts*, 1994.

[21] L. Juszczyk and S. Dustdar. Script-based generation of dynamic testbeds for soa. In *8th IEEE International Conference on Web Services (ICWS)*, Miami, Florida, USA, July 2010.

[22] D. Lübke. Unit Testing BPEL Compositions. In L. Baresi and E. D. Nitto, editors, *Test and Analysis of Service-oriented Systems*, pages 149–171. Springer, 2007. ISBN 978-3-540-72911-2.

[23] J. Lenhard, A. Schönberger, and G. Wirtz. Edit Distance-Based Pattern Support Assessment of Orchestration Languages. In *On the Move 2011 Confederated International Conferences: CoopIS, IS, DOA and ODBASE,*, Hersonissos, 2011.

[24] N. Lohmann. A Feature-Complete Petri Net Semantics for WS-BPEL 2.0 and its Compiler BPEL2oWFN. Informatik-Berichte 212, Humboldt-Universität zu Berlin, August 2007.

[25] N. Lohmann and J. Kleine. Fully-automatic Translation of Open Workflow Net Models into Simple Abstract BPEL Processes. In *Modellierung*, volume P-127 of *LNI*, pages 57–72. GI, 2008.

[26] A. P. Mathur. *Foundations of Software Testing*. Dorling Kindersley, 2009. ISBN-13: 978-81-317-1660-1.

[27] OASIS. *WS-Security Core Specification 1.1*, 2006.

[28] OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.

[29] 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 *International Conference on Web Services (ICWS)*, pages 285–292, 2006.

[30] M. P. Papazoglou and D. Georgakopoulos. Service-oriented Computing. *Communications of the ACM*, 46(10):24–28, October 2003.

[31] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, October 2003.

[32] A. Schönberger. The CHORCH B2Bi Approach: Performing ebBP Choreographies as Distributed BPEL Orchestrations. In *International Workshop on Services Computing for B2B (SC4B2B)*, Miami, Florida, USA, July 2010.

[33] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek. Conformance Checking of Service Behavior. *ACM Transactions on Internet Technology (TOIT)*, 8(3), May 2008.

[34] 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 *Web Services and Formal Methods, Forth International Workshop, WS-FM*, pages 139–153, September 2007.

[35] W. M. P. van der Aalst, A. Mooij, C. Stahl, and K. Wolf. Service Interaction: Patterns, Formalization, and Analysis. In *Formal Methods for Web Services*, volume 5569 of *LNCS*, pages 42–88. Springer Berlin/ Heidelberg, 2009. ISBN: 978-3-642-01917-3.

[36] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases, Springer*, 14(1):5–51, July 2003.

[37] W3C. *XML Path Language (XPath) Version 1.0*, November 1999. v1.0.

[38] W3C. *XSL Transformations (XSLT) Version 1.0*, November 1999.

[39] W3C. *Simple Object Access Protocol (SOAP) 1.1*, 2000.

[40] W3C. *Web Services Description Language (WSDL) 1.1*, March 2001.

[41] W3C. *Web Services Architecture*, February 2004.

[42] W3C. *XML Schema Part 1: Structures Second Edition*, October 2004.

[43] W3C. *XML Schema Part 2: Datatypes Second Edition*, October 2004.

[44] W3C. *Web Services Addressing 1.0 - Core*, 2006.

[45] I. Weber, J. Haller, and J. Mulle. Automated Derivation of Executable Business Processes from Choreographies in Virtual Organisations. *International Journal of Business Process Integration and Management*, 3:85–95, 2008.

[46] Z. Zakaria, R. Atan, A. Ghani, and N. Sani. Unit Testing Approaches for BPEL: A systematic Review. In *Asia-Pacific Software Engineering Conference*, pages 316–322, Penang, Malaysia, December 2009. IEEE.

A Compact Result Tables

A.1 basic-activities

basic-activities	bpelg	ode	openesb	orchestra	petalsesb
Assign-Copy-DoXslTransform	-	-	-	-	-
Assign-Copy-DoXslTransform-InvalidSourceFault	-	-	-	-	-
Assign-Copy-DoXslTransform-SubLanguageExecutionFault	-	-	-	-	-
Assign-Copy-DoXslTransform-XsltStylesheetNotFound	+	-	-	-	-
Assign-Copy-GetVariableProperty	+	+	+	+	-
Assign-Copy-IgnoreMissingFromData	+	+	+	+	+
Assign-Copy-KeepSrcElementName	+	-	-	-	-
Assign-Copy-Query	+	+	+	+	-
Assign-Empty	-	-	-	-	-
Assign-Expression-From	+	+	+	+	+
Assign-Expression-To	+	+	+	+	+
Assign-Literal	+	+	+	+	+
Assign-PartnerLink	+	-	-	-	-
Assign-PartnerLink-UnsupportedReference	-	-	+	-	-
Assign-Property	+	+	+	-	+
Assign-SelectionFailure	+	+	+	-	-
Assign-Validate	+	-	+	-	-
Assign-VariablesUnchangedInspiteOfFault	+	+	+	+	+
Empty	+	+	+	+	+
Exit	+	+	+	+	+
ExtensionActivity-MustUnderstand	-	-	-	+	+
ExtensionActivity-NoMustUnderstand	+	+	+	-	-
Invoke-Async	+	+	+	+	+
Invoke-Catch	-	-	-	+	-
Invoke-CatchAll	+	-	-	+	+
Invoke-CompensationHandler	+	+	-	-	-

basic-activities	bpelg	ode	openesb	orchestra	petalsesb
Invoke-Correlation-InitAsync	+	-	-	-	-
Invoke-Correlation-InitSync	+	+	-	+	-
Invoke-Correlation-Pattern-InitAsync	+	-	+	-	-
Invoke-Correlation-Pattern-InitSync	+	+	-	-	-
Invoke-Empty	-	+	-	+	-
Invoke-FromParts	+	-	-	+	-
Invoke-Sync	+	+	+	+	+
Invoke-Sync-Fault	-	-	-	-	+
Invoke-ToParts	+	-	-	+	-
Receive	+	+	+	+	-
Receive-AmbiguousReceiveFault	+	-	-	-	-
Receive-ConflictingReceiveFault	-	-	-	-	-
Receive-Correlation-InitAsync	+	-	-	-	-
Receive-Correlation-InitSync	+	-	-	-	-
ReceiveReply	+	+	+	+	+
ReceiveReply-ConflictingRequestFault	-	-	-	-	-
ReceiveReply-Correlation-InitAsync	+	-	+	-	-
ReceiveReply-Correlation-InitSync	+	+	-	-	-
ReceiveReply-CorrelationViolation-Join	-	-	-	-	-
ReceiveReply-CorrelationViolation-No	-	-	+	-	-
ReceiveReply-CorrelationViolation-Yes	+	-	-	-	-
ReceiveReply-Fault	+	+	+	+	-
ReceiveReply-FromParts	+	-	-	+	-
ReceiveReply-MessageExchanges	+	+	+	+	-
ReceiveReply-ToParts	+	-	-	+	-
Rethrow	+	+	+	-	-
Rethrow-FaultData	+	+	-	-	-
Rethrow-FaultDataUnmodified	+	+	-	-	-
Throw	+	+	+	-	-
Throw-CustomFault	+	+	+	-	-
Throw-CustomFaultInWsdl	+	+	+	-	-

basic-activities	bpelg	ode	openesb	orchestra	petalsesb
Throw-FaultData	+	+	-	-	-
Throw-WithoutNamespace	+	+	+	-	-
Validate	+	-	+	-	-
Validate-InvalidVariables	+	-	+	-	-
Variables-DefaultInitialization	+	-	-	+	+
Variables-UninitializedVariableFault-Invoke	+	+	+	-	-
Variables-UninitializedVariableFault-Reply	+	+	+	-	-
Wait-For	-	+	+	+	-
Wait-For-InvalidExpressionValue	+	+	+	-	-
Wait-Until	-	+	+	+	+

A.2 scopes

scopes	bpelg	ode	openesb	orchestra	petalsesb
MissingReply	+	-	-	-	-
MissingRequest	+	-	-	-	-
Scope-Compensate	+	+	+	-	-
Scope-CompensateScope	+	+	+	-	-
Scope-ComplexCompensation	+	+	+	+	-
Scope-CorrelationSets-InitAsync	+	-	+	-	-
Scope-CorrelationSets-InitSync	+	+	-	-	-
Scope-EventHandlers-InitAsync	+	-	+	-	-
Scope-EventHandlers-InitSync	+	+	-	+	-
Scope-EventHandlers-OnAlarm-For	-	+	+	+	-
Scope-EventHandlers-OnAlarm-RepeatEvery	-	+	+	+	-
Scope-EventHandlers-OnAlarm-RepeatEvery-For	-	+	+	-	-
Scope-EventHandlers-OnAlarm-RepeatEvery-Until	-	+	+	-	-
Scope-EventHandlers-OnAlarm-Until	-	+	+	+	-
Scope-EventHandlers-Parts	+	-	-	-	-
Scope-ExitOnStandardFault	+	-	+	+	+

scopes	bpelg	ode	openesb	orchestra	petalsesb
Scope-ExitOnStandardFault-JoinFailure	+	+	+	-	-
Scope-FaultHandlers	+	+	+	+	+
Scope-FaultHandlers-CatchAll	+	+	+	+	+
Scope-FaultHandlers-CatchOrder	+	+	+	-	+
Scope-FaultHandlers-FaultElement	+	+	+	-	+
Scope-FaultHandlers-FaultMessageType	+	+	+	-	+
Scope-FaultHandlers-VariableData	+	+	+	-	-
Scope-Isolated	+	+	+	-	-
Scope-MessageExchanges	+	+	+	+	-
Scope-PartnerLinks	+	+	+	+	-
Scope-RepeatableConstructCompensation	+	-	+	+	-
Scope-RepeatedCompensation	+	+	+	-	-
Scope-TerminationHandlers	-	-	+	+	-
Scope-TerminationHandlers-FaultNotPropagating	-	-	+	+	-
Scope-Variables	+	+	+	+	-
Scope-Variables-Overwriting	+	+	+	+	-

A.3 structured-activities

structured-activities	bpelg	ode	openesb	orchestra	petalsesb
Flow	+	+	+	+	-
Flow-BoundaryLinks	+	+	-	+	-
Flow-GraphExample	+	+	-	+	-
Flow-Links	+	+	-	+	-
Flow-Links-JoinCondition	+	+	+/-	+/-	-
Flow-Links-JoinFailure	+	+	-	-	-
Flow-Links-ReceiveCreatingInstances	+	+	+	+	-
Flow-Links-SuppressJoinFailure	+	+	-	+	-
Flow-Links-TransitionCondition	+	+	+/-	+	-
Flow-ReceiveCreatingInstances	-	-	-	+	-

structured-activities	bpelg	ode	openesb	orchestra	petalsesb
ForEach	+	+	+	-	+
ForEach-CompletionCondition	+	-	+	-	-
ForEach-CompletionCondition-NegativeBranches	-	-	+	-	-
ForEach-CompletionCondition-Parallel	+	-	+	-	-
ForEach-CompletionCondition-SuccessfulBranchesOnly	+	-	+	-	-
ForEach-CompletionConditionFailure	+	-	+	-	-
ForEach-DuplicateCounterVariable	-	+	-	+	+
ForEach-NegativeStartCounter	+	-	+	-	-
ForEach-NegativeStopCounter	+	-	+	-	-
ForEach-Parallel	+	+	+	-	+
ForEach-TooLargeStartCounter	-	-	-	-	-
If	+	+	+	+	+
If-Else	+	+	+	+	+
If-ElseIf	+	+	+	+	+
If-ElseIf-Else	+	+	+	+	+
If-InvalidExpressionValue	+	-	-	-	-
Pick-Correlations-InitAsync	+	-	+	-	-
Pick-Correlations-InitSync	+	+	-	+	-
Pick-CreateInstance	+	+	+	+	+
Pick-OnAlarm-For	-	+	+	+	-
Pick-OnAlarm-Until	-	+	+	+	-
RepeatUntil	+	+	+	+	-
RepeatUntilEquality	+	-	+	+	-
Sequence	+	+	+	+	+
While	+	+	+	+	+

B Test Descriptions and Results

<table>
<tr><td>Process name</td><td>Assign-Copy-DoXslTransform</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate assign that uses the doXslTransform function.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 9: Assign-Copy-DoXslTransform Test

<table>
<tr><td>Process name</td><td>Assign-Copy-DoXslTransform-InvalidSourceFault</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate assign that uses the doXslTransform function without a proper source for the script.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: xsltInvalidSource</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 10: Assign-Copy-DoXslTransform-InvalidSourceFault Test

<table>
<tr><td>Process name</td><td>Assign-Copy-DoXslTransform-SubLanguageExecutionFault</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate assign that uses the doXslTransform function, but where the actual stylesheet has errors.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: subLanguageExecutionFault</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 11: Assign-Copy-DoXslTransform-SubLanguageExecutionFault Test

Process name	Assign-Copy-DoXslTransform-XsltStylesheetNotFound
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign that uses the doXslTransform function, but where the stylesheet does not exist.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: xsltStylesheetNotFound
Support	bpel-g: +; ODE: -; openESB: -; Orchestra: -; PetalsESB: -

Table 12: Assign-Copy-DoXslTransform-XsltStylesheetNotFound Test

Process name	Assign-Copy-GetVariableProperty
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign that uses the getVariableProperty function.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 5
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: -

Table 13: Assign-Copy-GetVariableProperty Test

Process name	Assign-Copy-IgnoreMissingFromData
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign with a copy that has ignoreMissingFromData set to yes and contains a from element with an erroneous xpath statement. Therefore, the assign should be ignored.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: -1
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: +

Table 14: Assign-Copy-IgnoreMissingFromData Test

Process name	Assign-Copy-KeepSrcElementName
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign with a copy that has keepSrcElementName set to yes. This should trigger a fault.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: mismatchedAssignmentFailure
Support	bpel-g: +; ODE: -; openESB: -; Orchestra: -; PetalsESB: -

Table 15: Assign-Copy-KeepSrcElementName Test

Process name	Assign-Copy-Query
Activities and configuration	assign query receive reply sequence
Description	A process with a receive-reply pair with an intermediate assign that uses a query in a from element.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 5
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: -

Table 16: Assign-Copy-Query Test

Process name	Assign-Empty
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign that uses empty to and from elements.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 5
Support	bpel-g: -; ODE: -; openESB: -; Orchestra: -; PetalsESB: -

Table 17: Assign-Empty Test

Process name	Assign-Expression-From
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign that uses an expression in a from element.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 5
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: +

Table 18: Assign-Expression-From Test

Process name	Assign-Expression-To
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign that uses an expression in a to element.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 5
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: +

Table 19: Assign-Expression-To Test

<table>
<tr><td>Process name</td><td>Assign-Literal</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate assign that copies a literal.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 20: Assign-Literal Test

<table>
<tr><td>Process name</td><td>Assign-PartnerLink</td></tr>
<tr><td>Activities and configuration</td><td>addr:EndpointReference assign invoke receive reply sequence sref:service-ref</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate assign that assigns a WS-A EndpointReference to a partnerLink which is used in a subsequent invoke.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 0</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 21: Assign-PartnerLink Test

<table>
<tr><td>Process name</td><td>Assign-PartnerLink-UnsupportedReference</td></tr>
<tr><td>Activities and configuration</td><td>assign invoke receive reply sequence sref:service-ref</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate assign that assigns a bogus refernce to a partnerLink which is used in a subsequent invoke. The refernce scheme should not be supported by any engine and fail with a corresponding fault.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: unsupportedReference</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>+</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 22: Assign-PartnerLink-UnsupportedReference Test

<table>
<tr><td>Process name</td><td>Assign-Property</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate assign that copies from a property instead of a variable.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>+</td></tr></table></td></tr>
</table>

Table 23: Assign-Property Test

Process name	Assign-SelectionFailure
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign that uses a from that retuns zero nodes. This should trigger a selectionFailure.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: selectionFailure</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table>

Table 24: Assign-SelectionFailure Test

Process name	Assign-Validate
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair with an intermediate assign that has validate set to yes. The assign copies to a variable that represents a month and the validation should fail for values not in the range of one to twelve.
Test case: Input Value 13 should return validation fault	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>13</td><td>synchronous</td><td>fault: invalidVariables</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>+</td><td>-</td><td>-</td></tr></table>

Table 25: Assign-Validate Test

Process name	Assign-VariablesUnchangedInspiteOfFault
Activities and configuration	assign catchAll faultHandlers receive reply sequence
Description	A receive-reply pair with two intermediate assigns, the second of which produces a fault that is handled by the process-level faultHandler to send the response. Because of the fault, the second assign should have no impact on the response.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: -1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table>

Table 26: Assign-VariablesUnchangedInspiteOfFault Test

<table>
<tr><td>Process name</td><td>Empty</td></tr>
<tr><td>Activities and configuration</td><td>assign empty receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate empty.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 27: Empty Test

<table>
<tr><td>Process name</td><td>Exit</td></tr>
<tr><td>Activities and configuration</td><td>assign exit receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate exit. There should not be a normal response.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>Exit</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 28: Exit Test

<table>
<tr><td>Process name</td><td>ExtensionActivity-MustUnderstand</td></tr>
<tr><td>Activities and configuration</td><td>assign extensionActivity extensions receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an extensionActivity from an extension that has mustUnderstand set to no. The process defnition must be rejected according to Sec. 14.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>NotDeployable</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 29: ExtensionActivity-MustUnderstand Test

Process name	ExtensionActivity-NoMustUnderstand
Activities and configuration	assign extensionActivity extensions receive reply sequence
Description	A receive-reply pair with an extensionActivity from an extension that has mustUnderstand set to no. The activity should be treated as an empty according to Sec. 10.9.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 5
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 30: ExtensionActivity-NoMustUnderstand Test

Process name	Invoke-Async
Activities and configuration	assign invoke receive reply sequence
Description	A receive-reply pair with an intermediate asynchronous invoke.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 5
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: +

Table 31: Invoke-Async Test

Process name	Invoke-Catch
Activities and configuration	assign catch invoke receive reply sequence
Description	A receive-reply pair with an intermediate invoke that results in a fault for certain input, but catches that fault and replies.
Test case: Good-Case	input: -5; operation: synchronous; assertions: output: 0
Support	bpel-g: -; ODE: -; openESB: -; Orchestra: +; PetalsESB: -

Table 32: Invoke-Catch Test

Process name	Invoke-CatchAll
Activities and configuration	assign catchAll invoke receive reply sequence
Description	A receive-reply pair with an intermediate invoke that results in a fault for certain input, but catches all faults and replies.
Test case: Enter-CatchAll	input: -5; operation: synchronous; assertions: output: 0
Support	bpel-g: +; ODE: -; openESB: -; Orchestra: +; PetalsESB: +

Table 33: Invoke-CatchAll Test

Process name	Invoke-CompensationHandler
Activities and configuration	assign catchAll compensate compensationHandler faultHandlers invoke receive reply sequence throw
Description	A receive-reply pair combined with an invoke that has a compensationHandler, followed by a throw. The fault is caught by the process-level faultHandler. That faultHandler triggers the compensationHandler of the invoke which contains the reply.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 0</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>-</td><td>-</td><td>-</td></tr></table>

Table 34: Invoke-CompensationHandler Test

Process name	Invoke-Correlation-InitAsync
Activities and configuration	assign correlationSets invoke receive reply sequence
Description	An asynchronous receive that initiates a correlationSet used by a subsequent invoke which is thereafter followed by receive-reply pair that also uses the correlationSet.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>asynchronous</td><td></td></tr><tr><td>wait for</td><td>1000 ms</td><td></td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table>

Table 35: Invoke-Correlation-InitAsync Test

Process name	Invoke-Correlation-InitSync
Activities and configuration	assign correlationSets invoke receive reply sequence
Description	A synchronous receive that initiates a correlationSet used by a subsequent invoke which is thereafter followed by receive-reply pair that also uses the correlationSet.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 0</td></tr><tr><td>wait for</td><td>1000 ms</td><td></td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>-</td><td>+</td><td>-</td></tr></table>

Table 36: Invoke-Correlation-InitSync Test

<table>
<tr><td>Process name</td><td>Invoke-Correlation-Pattern-InitAsync</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets invoke receive reply sequence</td></tr>
<tr><td>Description</td><td>An asynchronous receive that initiates a correlationSet used by a subsequent invoke that also uses a request-response pattern and is thereafter followed by receive-reply pair that also uses the correlationSet.</td></tr>
<tr><td>Test case: Good-Case</td><td>
<table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>1</td><td>asynchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td>output: 1</td></tr>
</table>
</td></tr>
<tr><td>Support</td><td>
<table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>-</td><td>+</td><td>-</td><td>-</td></tr>
</table>
</td></tr>
</table>

Table 37: Invoke-Correlation-Pattern-InitAsync Test

<table>
<tr><td>Process name</td><td>Invoke-Correlation-Pattern-InitSync</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets invoke receive reply sequence</td></tr>
<tr><td>Description</td><td>A synchronous receive that initiates a correlationSet used by a subsequent invoke that also uses a request-response pattern and is thereafter followed by receive-reply pair that also uses the correlationSet.</td></tr>
<tr><td>Test case: Good-Case</td><td>
<table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>1</td><td>synchronous</td><td>output: 0</td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td>output: 1</td></tr>
</table>
</td></tr>
<tr><td>Support</td><td>
<table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>+</td><td>-</td><td>-</td><td>-</td></tr>
</table>
</td></tr>
</table>

Table 38: Invoke-Correlation-Pattern-InitSync Test

<table>
<tr><td>Process name</td><td>Invoke-Empty</td></tr>
<tr><td>Activities and configuration</td><td>assign invoke receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate invoke of an operation that has no message associated with it. No definition of inputVariable or outputVariable is required.</td></tr>
<tr><td>Test case: Good-Case</td><td>
<table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>5</td><td>synchronous</td><td>output: 5</td></tr>
</table>
</td></tr>
<tr><td>Support</td><td>
<table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>-</td><td>+</td><td>-</td><td>+</td><td>-</td></tr>
</table>
</td></tr>
</table>

Table 39: Invoke-Empty Test

<table>
<tr><td>Process name</td><td>Invoke-FromParts</td></tr>
<tr><td>Activities and configuration</td><td>assign fromParts invoke receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate synchronous invoke that uses the fromParts syntax.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 40: Invoke-FromParts Test

<table>
<tr><td>Process name</td><td>Invoke-Sync</td></tr>
<tr><td>Activities and configuration</td><td>assign invoke receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate synchronous invoke.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 41: Invoke-Sync Test

<table>
<tr><td>Process name</td><td>Invoke-Sync-Fault</td></tr>
<tr><td>Activities and configuration</td><td>assign invoke receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate synchronous invoke that should trigger a fault.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>-5</td><td>synchronous</td><td>fault: CustomFault</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>+</td></tr></table></td></tr>
</table>

Table 42: Invoke-Sync-Fault Test

<table>
<tr><td>Process name</td><td>Invoke-ToParts</td></tr>
<tr><td>Activities and configuration</td><td>assign invoke receive reply sequence toParts</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate synchronous invoke that uses the toParts syntax.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 43: Invoke-ToParts Test

Process name	Receive
Activities and configuration	receive
Description	A single asynchronous receive.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>asynchronous</td><td></td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table>

Table 44: Receive Test

<table>
<tr><td>Process name</td><td>Receive-AmbiguousReceiveFault</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets flow receive reply sequence</td></tr>
<tr><td>Description</td><td>An asynchronous receive that initiates two correlationSets, followed by a flow with two sequences that contain synchronous receive-reply pairs for the same operation but differnet correlationSets. Should trigger an ambiguousReceive fault.</td></tr>
<tr><td>Test case: Good-Case</td><td><table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>1</td><td>asynchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td>fault: ambiguousReceive</td></tr>
</table></td></tr>
<tr><td>Support</td><td><table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table></td></tr>
</table>

Table 45: Receive-AmbiguousReceiveFault Test

<table>
<tr><td>Process name</td><td>Receive-ConflictingReceiveFault</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets flow receive reply sequence</td></tr>
<tr><td>Description</td><td>An asynchronous receive that iniates a correlationSet, followed by a flow with two sequences that contain synchronous receive-reply pair for the same operation and correlationSet. Should trigger a conflictingReceive fault.</td></tr>
<tr><td>Test case: Good-Case</td><td><table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>1</td><td>synchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td>fault: conflictingReceive</td></tr>
</table></td></tr>
<tr><td>Support</td><td><table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table></td></tr>
</table>

Table 46: Receive-ConflictingReceiveFault Test

<table>
<tr><td>Process name</td><td>Receive-Correlation-InitAsync</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets receive reply sequence</td></tr>
<tr><td>Description</td><td>Two asynchronous receives, followed by a receive-reply pair, and bound to a single correlationSet.</td></tr>
<tr><td>Test case: Good-Case</td><td><table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>1</td><td>asynchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>asynchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td>output: 1</td></tr>
</table></td></tr>
<tr><td>Support</td><td><table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table></td></tr>
</table>

Table 47: Receive-Correlation-InitAsync Test

<table>
<tr><td>Process name</td><td>Receive-Correlation-InitSync</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets receive reply sequence</td></tr>
<tr><td>Description</td><td>One synchronous receive, one asynchronous receive, followed by a receive-reply pair, and bound to a single correlationSet.</td></tr>
<tr><td>Test case: Good-Case</td><td><table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>1</td><td>synchronous</td><td>output: 0</td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>asynchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td>output: 1</td></tr>
</table></td></tr>
<tr><td>Support</td><td><table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table></td></tr>
</table>

Table 48: Receive-Correlation-InitSync Test

<table>
<tr><td>Process name</td><td>ReceiveReply</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence</td></tr>
<tr><td>Description</td><td>A simple receive-reply pair.</td></tr>
<tr><td>Test case: Good-Case</td><td><table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>5</td><td>synchronous</td><td>output: 5</td></tr>
</table></td></tr>
<tr><td>Support</td><td><table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr>
</table></td></tr>
</table>

Table 49: ReceiveReply Test

<table>
<tr><td>Process name</td><td>ReceiveReply-ConflictingRequestFault</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets messageExchanges receive reply sequence while</td></tr>
<tr><td>Description</td><td>A synchronous interaction, followed by intermediate while that subsequently enables multiple receives that correspond to a single synchronous message exchange. Should trigger a conflictingRequest fault.</td></tr>
<tr><td>Test case: Good-Case</td><td>
<table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>1</td><td>synchronous</td><td>output: 1</td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>1</td><td>synchronous</td><td>fault: conflictingRequest</td></tr>
</table>
</td></tr>
<tr><td>Support</td><td>
<table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table>
</td></tr>
</table>

Table 50: ReceiveReply-ConflictingRequestFault Test

<table>
<tr><td>Process name</td><td>ReceiveReply-Correlation-InitAsync</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets receive reply sequence</td></tr>
<tr><td>Description</td><td>An asynchronous receive that initiates a correlationSet followed by a receive-reply pair that uses this set.</td></tr>
<tr><td>Test case: Good-Case</td><td>
<table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>5</td><td>asynchronous</td><td></td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>5</td><td>synchronous</td><td>output: 5</td></tr>
</table>
</td></tr>
<tr><td>Support</td><td>
<table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>-</td><td>+</td><td>-</td><td>-</td></tr>
</table>
</td></tr>
</table>

Table 51: ReceiveReply-Correlation-InitAsync Test

<table>
<tr><td>Process name</td><td>ReceiveReply-Correlation-InitSync</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets receive reply sequence</td></tr>
<tr><td>Description</td><td>A synchronous recieve that initiates a correlationSet followed by a receive-reply pair that uses this set.</td></tr>
<tr><td>Test case: Good-Case</td><td>
<table>
<tr><td>input</td><td>operation</td><td>assertions</td></tr>
<tr><td>5</td><td>synchronous</td><td>output: 0</td></tr>
<tr><td>wait for</td><td>1000 ms</td><td></td></tr>
<tr><td>5</td><td>synchronous</td><td>output: 5</td></tr>
</table>
</td></tr>
<tr><td>Support</td><td>
<table>
<tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr>
<tr><td>+</td><td>+</td><td>-</td><td>-</td><td>-</td></tr>
</table>
</td></tr>
</table>

Table 52: ReceiveReply-Correlation-InitSync Test

<table>
<tr><td>Process name</td><td>ReceiveReply-CorrelationViolation-Join</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets invoke receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair that initates a correlationSet with an intermediate invoke that tries to join the correlationSet. The join operation should only work if the correlationSet was initiate with a certain value.</td></tr>
<tr><td>Test case: Good-Case-1</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: correlationViolation</td></tr></table></td></tr>
<tr><td>Test case: Good-Case-2</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 2</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 53: ReceiveReply-CorrelationViolation-Join Test

<table>
<tr><td>Process name</td><td>ReceiveReply-CorrelationViolation-No</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair that uses an uninitiated correlationSet and sets initiate to no. Should trigger a correlationViolation fault.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: correlationViolation</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>+</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 54: ReceiveReply-CorrelationViolation-No Test

<table>
<tr><td>Process name</td><td>ReceiveReply-CorrelationViolation-Yes</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets receive reply sequence</td></tr>
<tr><td>Description</td><td>Two subsequent receive-reply pairs which share a correlationSet and where both receives have initiate set to yes.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr><tr><td>wait for</td><td>1000 ms</td><td></td></tr><tr><td>1</td><td>synchronous</td><td>fault: correlationViolation</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 55: ReceiveReply-CorrelationViolation-Yes Test

Process name	ReceiveReply-Fault
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair replies with a fault instead of a variable.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: syncFault
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: -

Table 56: ReceiveReply-Fault Test

Process name	ReceiveReply-FromParts
Activities and configuration	assign fromParts receive reply sequence
Description	A receive-reply pair that uses the fromPart syntax instead of a variable.
Test case: Good-Case	input: 1; operation: synchronous; assertions: output: 1
Support	bpel-g: +; ODE: -; openESB: -; Orchestra: +; PetalsESB: -

Table 57: ReceiveReply-FromParts Test

Process name	ReceiveReply-MessageExchanges
Activities and configuration	assign messageExchanges receive reply sequence
Description	A simple receive-reply pair that uses a messageExchange.
Test case: Good-Case	input: 1; operation: synchronous; assertions: output: 1
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: -

Table 58: ReceiveReply-MessageExchanges Test

Process name	ReceiveReply-ToParts
Activities and configuration	assign receive reply sequence toParts
Description	A receive-reply pair that uses the toPart syntax instead of a variable.
Test case: Good-Case	input: 1; operation: synchronous; assertions: output: 1
Support	bpel-g: +; ODE: -; openESB: -; Orchestra: +; PetalsESB: -

Table 59: ReceiveReply-ToParts Test

Process name	Rethrow
Activities and configuration	assign catchAll faultHandlers receive rethrow sequence throw
Description	A receive activity with an intermediate throw and a fault handler with a catchAll. The fault handler rethrows the fault.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: completionConditionFailure
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

input	operation	assertions
1	synchronous	fault: completionConditionFailure

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	+	-	-

Table 60: Rethrow Test

Process name	Rethrow-FaultData
Activities and configuration	assign catch faultHandlers receive rethrow scope sequence throw
Description	A receive activity with an intermediate throw that uses a faultVariable. A fault handler catches and rethrows the fault. The fault should be the response along with the data.
Test case: Good-Case	
Support	

input	operation	assertions
1	synchronous	output: 1; fault: completionConditionFailure

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	-	-	-

Table 61: Rethrow-FaultData Test

Process name	Rethrow-FaultDataUnmodified
Activities and configuration	assign catch faultHandlers receive rethrow scope sequence throw
Description	A receive activity with an intermediate throw that uses a faultVariable. A fault handler catches the fault, changes the data, and rethrows the fault. The fault should be the response with unchanged data.
Test case: Good-Case	
Support	

input	operation	assertions
1	synchronous	output: 1; fault: completionConditionFailure

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	-	-	-

Table 62: Rethrow-FaultDataUnmodified Test

Process name	Throw
Activities and configuration	assign receive reply sequence throw
Description	A receive-reply pair with an intermediate throw. The response should a soap fault containing the bpel fault.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: completionConditionFailure</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table>

Table 63: Throw Test

Process name	Throw-CustomFault
Activities and configuration	receive sequence throw
Description	A receive-reply pair with an intermediate throw that throws a custom fault that undefined in the given namespace. The response should be a soap fault containing the custom fault.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: testFault</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table>

Table 64: Throw-CustomFault Test

Process name	Throw-CustomFaultInWsdl
Activities and configuration	assign receive sequence throw
Description	A receive-reply pair with an intermediate throw that throws a custom fault defined in the myRole WSDL. The response should be a soap fault containing the custom fault.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: syncFault
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 65: Throw-CustomFaultInWsdl Test

Process name	Throw-FaultData
Activities and configuration	assign receive reply sequence throw
Description	A receive-reply pair with an intermediate throw that also uses a faultVariable. The content of the faultVariable should be contained in the response.
Test case: Good-Case	input: 1; operation: synchronous; assertions: output: 1; fault: completionConditionFailure
Support	bpel-g: +; ODE: +; openESB: -; Orchestra: -; PetalsESB: -

Table 66: Throw-FaultData Test

Process name	Throw-WithoutNamespace
Activities and configuration	assign receive reply sequence throw
Description	A receive-reply pair with an intermediate throw that uses a bpel fault without explicitly using the bpel namespace. The respone should be a soap fault containing the bpel fault.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: completionConditionFailure
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 67: Throw-WithoutNamespace Test

Process name	Validate
Activities and configuration	assign receive reply sequence validate
Description	A receive-reply pair with an intermediate variable validation. The variable to be validated describes a month, so only values in the range of 1 and 12 should validate successfully.
Test case: Input Value 13 should return validation fault	input: 13; operation: synchronous; assertions: fault: invalidVariables
Support	bpel-g: +; ODE: -; openESB: +; Orchestra: -; PetalsESB: -

Table 68: Validate Test

Process name	Validate-InvalidVariables
Activities and configuration	assign receive reply sequence validate
Description	A receive-reply pair with an intermediate variable validation. The variable to be validated is of type xs:int and xs:boolean is copied into it.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: invalidVariables
Support	bpel-g: +; ODE: -; openESB: +; Orchestra: -; PetalsESB: -

Table 69: Validate-InvalidVariables Test

Process name	Variables-DefaultInitialization
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair where the variable of the reply is assigned with a default value.
Test case: DefaultValue-10-Should-Be-Returned	input: 5; operation: synchronous; assertions: output: 10
Support	bpel-g: +; ODE: -; openESB: -; Orchestra: +; PetalsESB: +

Table 70: Variables-DefaultInitialization Test

Process name	Variables-UninitializedVariableFault-Invoke
Activities and configuration	invoke receive reply sequence
Description	A receive-reply pair with intermediate invoke. The inputVariable of the invoke is not initialized.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: uninitializedVariable
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 71: Variables-UninitializedVariableFault-Invoke Test

Process name	Variables-UninitializedVariableFault-Reply
Activities and configuration	receive reply sequence
Description	A receive-reply pair where the variable of the reply is not initialized.
Test case: Good-Case	input: 1; operation: synchronous; assertions: fault: uninitializedVariable
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 72: Variables-UninitializedVariableFault-Reply Test

<table>
<tr><td>Process name</td><td>Wait-For</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence wait</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate wait that pauses execution for five seconds.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 73: Wait-For Test

<table>
<tr><td>Process name</td><td>Wait-For-InvalidExpressionValue</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence wait</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate wait. The for element is assigned a value of xs:int, but only xs:duration is allowed.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>fault: invalidExpressionValue</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 74: Wait-For-InvalidExpressionValue Test

<table>
<tr><td>Process name</td><td>Wait-Until</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence wait</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate wait that pauses the execution until a date in the past. Therefore, the wait should complete instantly.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 75: Wait-Until Test

<table>
<tr><td>Process name</td><td>MissingReply</td></tr>
<tr><td>Activities and configuration</td><td>assign receive sequence</td></tr>
<tr><td>Description</td><td>A receive for a synchronous operation with no associated reply.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: missingReply</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 76: MissingReply Test

<table>
<tr><td>Process name</td><td>MissingRequest</td></tr>
<tr><td>Activities and configuration</td><td>assign messageExchanges receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive and a reply which belong to different messageExchanges. On the execution of the reply, a missingRequest fault should be thrown.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: missingRequest</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 77: MissingRequest Test

<table>
<tr><td>Process name</td><td>Scope-Compensate</td></tr>
<tr><td>Activities and configuration</td><td>assign catchAll compensate compensationHandler faultHandlers receive reply scope sequence throw</td></tr>
<tr><td>Description</td><td>A scope with a receive-reply pair where the reply is located in a compensationHandler. The scope is followed by a throw and the compensationHandler is invoked from the process-level faultHandler that catches the fault using compensate.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 78: Scope-Compensate Test

<table>
<tr><td>Process name</td><td>Scope-CompensateScope</td></tr>
<tr><td>Activities and configuration</td><td>assign catchAll compensateScope compensationHandler faultHandlers receive reply scope sequence throw</td></tr>
<tr><td>Description</td><td>A scope with a receive-reply pair where the reply is located in a compensationHandler. The scope is followed by a throw and the compensationHandler is invoked from the process-level faultHandler that catches the fault using compensateScope.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 79: Scope-CompensateScope Test

Process name	Scope-ComplexCompensation
Activities and configuration	assign catchAll compensate compensationHandler empty faultHandlers receive reply scope sequence throw
Description	Complex scope compensation test case that implements the scenario described in Sec. 12.4.2.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 3</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table>

Table 80: Scope-ComplexCompensation Test

Process name	Scope-CorrelationSets-InitAsync
Activities and configuration	assign correlationSets receive reply scope sequence
Description	A scope with an asynchronous receive which initiates the correlation set and a receive-reply pair, as well as a scope-level definition of a correlationSet that is used by the messaging activities.
Test case: Good-Case	input | operation | assertions 1 | asynchronous | 1 | synchronous | output: 2
Support	bpel-g: + ; ODE: - ; openESB: + ; Orchestra: - ; PetalsESB: -

Table 81: Scope-CorrelationSets-InitAsync Test

Process name	Scope-CorrelationSets-InitSync
Activities and configuration	assign correlationSets receive reply scope sequence
Description	A scope with two subsequent receive-reply pairs and a scope-level definition of a correlationSet that is used by the messaging activities.
Test case: Good-Case	input | operation | assertions 1 | synchronous | output: 1 1 | synchronous | output: 2
Support	bpel-g: + ; ODE: + ; openESB: - ; Orchestra: - ; PetalsESB: -

Table 82: Scope-CorrelationSets-InitSync Test

Process name	Scope-EventHandlers-InitAsync
Activities and configuration	assign correlationSets eventHandlers onEvent receive reply scope sequence wait
Description	An asynchronous receive followed by a wait and a process-level onMessage eventHandler. The receive initiates a correlationSet on which the onMessage correlates with a synchronous operation.
Test case: Good-Case	input / operation / assertions 5 / asynchronous / 5 / synchronous / output: 5
Support	bpel-g: + ; ODE: - ; openESB: + ; Orchestra: - ; PetalsESB: -

Table 83: Scope-EventHandlers-InitAsync Test

Process name	Scope-EventHandlers-InitSync
Activities and configuration	assign correlationSets eventHandlers onEvent receive reply scope sequence wait
Description	A receive-reply pair followed by a wait and a process-level onMessage eventHandler. The receive initiates a correlationSet on which the onMessage correlates with a synchronous operation.
Test case: Good-Case	input / operation / assertions 1 / synchronous / output: 1 wait for / 3000 ms / 1 / synchronous / output: 2
Support	bpel-g: + ; ODE: + ; openESB: - ; Orchestra: + ; PetalsESB: -

Table 84: Scope-EventHandlers-InitSync Test

Process name	Scope-EventHandlers-OnAlarm-For
Activities and configuration	assign eventHandlers onAlarm receive reply scope sequence wait
Description	A receive-reply pair and a process-level onAlarm eventHandler. The receive is followed by a wait that pauses execution for five seconds. The eventHandler waits for two seconds and replies to the receive.
Test case: Good-Case	input / operation / assertions 5 / synchronous / output: 5
Support	bpel-g: - ; ODE: + ; openESB: + ; Orchestra: + ; PetalsESB: -

Table 85: Scope-EventHandlers-OnAlarm-For Test

Process name	Scope-EventHandlers-OnAlarm-RepeatEvery
Activities and configuration	assign eventHandlers onAlarm receive repeatEvery reply scope sequence wait
Description	A receive-reply pair with an intermediate wait and a process-level onAlarm eventHandler. The eventHandler repeats execution every second and adds one to the final result. The intermediate wait pauses execution for 2.2 seconds, after which the current result is replied.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 2
Support	bpel-g: -; ODE: +; openESB: +; Orchestra: +; PetalsESB: -

Table 86: Scope-EventHandlers-OnAlarm-RepeatEvery Test

Process name	Scope-EventHandlers-OnAlarm-RepeatEvery-For
Activities and configuration	assign eventHandlers onAlarm receive repeatEvery reply scope sequence wait
Description	A receive-reply pair with an intermediate wait and a process-level onAlarm eventHandler. The eventHandler repeats execution every second and adds one to the final result. The repetition takes place after one second, so the handler should repeat exactly once. The intermediate wait pauses execution for 2.2 seconds, after which the current result is replied.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 1
Support	bpel-g: -; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 87: Scope-EventHandlers-OnAlarm-RepeatEvery-For Test

Process name	Scope-EventHandlers-OnAlarm-RepeatEvery-Until
Activities and configuration	assign eventHandlers onAlarm receive repeatEvery reply scope sequence wait
Description	A receive-reply pair with an intermediate wait and a process-level onAlarm eventHandler. The eventHandler repeats execution every second and adds one to the final result. The repetition takes place after a date in the past, so the handler should execute immediately. The intermediate wait pauses execution for 2.2 seconds, after which the current result is replied.
Test case: Good-Case	input: 5; operation: synchronous; assertions: output: 2
Support	bpel-g: -; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 88: Scope-EventHandlers-OnAlarm-RepeatEvery-Until Test

<table>
<tr><td>Process name</td><td>Scope-EventHandlers-OnAlarm-Until</td></tr>
<tr><td>Activities and configuration</td><td>assign eventHandlers onAlarm receive reply scope sequence wait</td></tr>
<tr><td>Description</td><td>A receive followed by a scope with an onAlarm eventHandler and a wait. The onAlarm waits until a date in the past and should therefore execute immediately. Its body contains the reply to the initial receive.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 89: Scope-EventHandlers-OnAlarm-Until Test

<table>
<tr><td>Process name</td><td>Scope-EventHandlers-Parts</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets eventHandlers fromParts onEvent receive reply scope sequence wait</td></tr>
<tr><td>Description</td><td>An asynchronous receive followed by a wait and a process-level onMessage eventHandler. The receive initiates a correlationSet on which the onMessage correlates with a synchronous operation. Furthermore, the onMessage uses the fromPart syntax.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>asynchronous</td><td></td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 90: Scope-EventHandlers-Parts Test

<table>
<tr><td>Process name</td><td>Scope-ExitOnStandardFault</td></tr>
<tr><td>Activities and configuration</td><td>assign receive reply sequence throw</td></tr>
<tr><td>Description</td><td>A scope with receive-reply pair and an intermediate throw. There is no faultHandler, but the exitOnStandardFault attribute of the scope is set to yes.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>Exit</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 91: Scope-ExitOnStandardFault Test

Process name	Scope-ExitOnStandardFault-JoinFailure
Activities and configuration	assign receive reply scope sequence throw
Description	A scope with a receive-reply pair and an intermediate throw that throws a joinFailure. There is no faultHandler, but the exitOnStandardFault attribute of the scope is set to yes. However, the exitOnStandardFault sematics do not apply to joinFailures.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: joinFailure</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table>

Table 92: Scope-ExitOnStandardFault-JoinFailure Test

Process name	Scope-FaultHandlers
Activities and configuration	assign catch faultHandlers receive reply scope sequence throw
Description	A scope with a receive followed by a intermediate throw. The fault that is thrown is caught by the scope-level faultHandler by its faultName. Inside this faultHandler is the reply to the initial receive.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table>

Table 93: Scope-FaultHandlers Test

Process name	Scope-FaultHandlers-CatchAll
Activities and configuration	assign catchAll faultHandlers receive reply scope sequence throw
Description	A scope with a receive followed by a intermediate throw. The fault that is thrown is caught by the scope-level catchAll faultHandler. Inside this faultHandler is the reply to the initial receive.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table>

Table 94: Scope-FaultHandlers-CatchAll Test

Process name	Scope-FaultHandlers-CatchOrder
Activities and configuration	assign catch catchAll empty faultHandlers receive reply scope sequence throw
Description	A scope with a receive followed by a intermediate throw. The scope is associated with mulitple faultHandlers. A specific one of these should catch the fault and only inside this faultHandler is the reply to the initial receive. The process is adapted from the example in Spec. 12.5.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>+</td></tr></table>

Table 95: Scope-FaultHandlers-CatchOrder Test

Process name	Scope-FaultHandlers-FaultElement
Activities and configuration	assign catch faultHandlers receive reply scope sequence throw
Description	A scope with a receive followed by a intermediate throw. The fault that is thrown is caught by the scope-level faultHandler that uses a faultVariable and faultElement configuration. Inside this faultHandler is the reply to the initial receive.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>+</td></tr></table>

Table 96: Scope-FaultHandlers-FaultElement Test

Process name	Scope-FaultHandlers-FaultMessageType
Activities and configuration	assign catch faultHandlers receive reply scope sequence throw
Description	A scope with a receive followed by a intermediate throw. The fault that is thrown is caught by the scope-level faultHandler that uses a faultVariable and faultMessageType configuration. Inside this faultHandler is the reply to the initial receive.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 5</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>+</td></tr></table>

Table 97: Scope-FaultHandlers-FaultMessageType Test

Process name	Scope-FaultHandlers-VariableData
Activities and configuration	assign catch faultHandlers receive reply scope sequence throw
Description	A scope with a receive followed by a intermediate throw. The fault that is thrown is caught by the scope-level faultHandler that uses a faultVariable and faultMessage configuration. Inside this faultHandler is the reply to the initial receive and the data replied is the content of the faultVariable.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 0</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table>

Table 98: Scope-FaultHandlers-VariableData Test

Process name	Scope-Isolated
Activities and configuration	assign flow receive reply scope sequence
Description	A receive-reply pair that encloses a flow with ten isolated scopes which all increment the result by one. As the scopes should not run in parallel, the outcome must be deterministic.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 11</td></tr><tr><td>4</td><td>synchronous</td><td>output: 14</td></tr><tr><td>123</td><td>synchronous</td><td>output: 133</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>-</td></tr></table>

Table 99: Scope-Isolated Test

Process name	Scope-MessageExchanges
Activities and configuration	assign messageExchanges receive reply scope sequence
Description	A scope with a receive-reply pair and a scope-level definition of messageExchanges.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table>

Table 100: Scope-MessageExchanges Test

Process name	Scope-PartnerLinks
Activities and configuration	assign invoke receive reply scope sequence
Description	A scope with a receive-reply pair and an intermediate invoke. The partnerLink which is invoked is defined at scope-level.
Test case: Good-Case	input: 1; operation: synchronous; assertions: output: 1
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: -

Table 101: Scope-PartnerLinks Test

Process name	Scope-RepeatableConstructCompensation
Activities and configuration	assign catchAll compensate compensationHandler faultHandlers receive reply scope sequence throw while
Description	A receive followed by a while that contains a scope with a compensationHandler. After the while comes a throw and its fault is caught by the process-level faultHandler. This faultHandler first invokes compensation of all scopes and the replies to the initial receive. The content of the reply depends on the execution of the compensationHandlers.
Test case: Good-Case	input: 3; operation: synchronous; assertions: output: 3
Support	bpel-g: +; ODE: -; openESB: +; Orchestra: +; PetalsESB: -

Table 102: Scope-RepeatableConstructCompensation Test

Process name	Scope-RepeatedCompensation
Activities and configuration	assign catchAll compensate compensationHandler faultHandlers receive reply scope sequence throw
Description	A scope with a receive-reply pair where the reply is located in a compensationHandler. The scope is followed by a throw. The process-level faultHandler that catches the fault contains two subsequent compensates the second of which should be treated as empty.
Test case: Good-Case	input: 1; operation: synchronous; assertions: output: 1
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: -; PetalsESB: -

Table 103: Scope-RepeatedCompensation Test

Process name	Scope-TerminationHandlers
Activities and configuration	assign catchAll empty faultHandlers flow receive reply scope sequence terminationHandler throw wait
Description	A scope with a receive-reply pair and a nested scope in between. That scope in turn contains a flow with two parallel scopes. Both scopes pause execution for a short period. The scope that resumes execution first throws a fault caught by the faultHandler of its parent scope. The should trigger the execution of the terminationHandler of its sibling scope.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: -1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>+</td><td>+</td><td>-</td></tr></table>

Table 104: Scope-TerminationHandlers Test

Process name	Scope-TerminationHandlers-FaultNotPropagating
Activities and configuration	assign catch empty faultHandlers flow receive reply rethrow scope sequence terminationHandler throw wait
Description	A scope with a receive-reply pair and a nested scope in between. That scope in turn contains a flow with two parallel scopes. Both scopes pause execution for a short period. The scope that resumes execution first throws a fault caught by the faultHandler of its parent scope. The should trigger the execution of the terminationHandler of its sibling scope. That terminationHandler also throws a fault which should not be propagated.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: -1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>+</td><td>+</td><td>-</td></tr></table>

Table 105: Scope-TerminationHandlers-FaultNotPropagating Test

Process name	Scope-Variables
Activities and configuration	assign receive reply scope sequence
Description	A scope with a receive-reply pair and an intermediate invoke. The partnerLink which is invoked is defined at scope-level.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table>

Table 106: Scope-Variables Test

Process name	Scope-Variables-Overwriting
Activities and configuration	assign receive reply scope sequence
Description	A scope with a receive-reply pair and another nested scope. The nested scope overwrites a variable of the parent scope. Child-level manipulation of this variable should not be visible at the parent scope.
Test case: Good-Case	input: 123; operation: synchronous; assertions: output: 3
Support	bpel-g: +; ODE: +; openESB: +; Orchestra: +; PetalsESB: -

Test case: Good-Case

input	operation	assertions
123	synchronous	output: 3

Support

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	+	+	-

Table 107: Scope-Variables-Overwriting Test

Process name	Flow
Activities and configuration	assign flow receive reply sequence
Description	A receive-reply pair with an intermediate flow that contains two assigns.

Test case: Good-Case

input	operation	assertions
5	synchronous	output: 7

Support

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	+	+	-

Table 108: Flow Test

Process name	Flow-BoundaryLinks
Activities and configuration	assign flow links receive reply sequence
Description	A receive-reply pair with an intermediate flow that contains an assign and a sequence with an assign, as well as a link pointing from the former to the later assign. That way the links crosses the boundary of a structured activity, the sequence.

Test case: Good-Case

input	operation	assertions
1	synchronous	output: 2

Support

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	-	+	-

Table 109: Flow-BoundaryLinks Test

Process name	Flow-GraphExample
Activities and configuration	assign correlationSets empty flow joinCondition links receive reply sequence
Description	An implementation of the flow graph process defined in Sec. 11.6.4.
Test case: Good-Case-1	input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: asynchronous input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: asynchronous
Test case: Good-Case-2	input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: asynchronous input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: asynchronous
Test case: Good-Case-3	input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: asynchronous input: 1, operation: asynchronous input: 1, operation: synchronous, assertions: output: 1
Test case: Good-Case-4	input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: asynchronous input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: asynchronous input: 1, operation: synchronous, assertions: output: 1
Support	bpel-g: +, ODE: +, openESB: -, Orchestra: +, PetalsESB: -

Test case: Good-Case-1

input	operation	assertions
1	synchronous	output: 1
1	synchronous	output: 1
1	asynchronous	
1	synchronous	output: 1
1	asynchronous	

Test case: Good-Case-2

input	operation	assertions
1	synchronous	output: 1
1	asynchronous	
1	synchronous	output: 1
1	synchronous	output: 1
1	asynchronous	

Test case: Good-Case-3

input	operation	assertions
1	synchronous	output: 1
1	synchronous	output: 1
1	asynchronous	
1	asynchronous	
1	synchronous	output: 1

Test case: Good-Case-4

input	operation	assertions
1	synchronous	output: 1
1	asynchronous	
1	synchronous	output: 1
1	asynchronous	
1	synchronous	output: 1

Support

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	-	+	-

Table 110: Flow-GraphExample Test

Process name	Flow-Links
Activities and configuration	assign flow links receive reply sequence
Description	A receive-reply pair with an intermediate flow that contains two assigns which have a precedence relationship between each other using links.

Test case: Good-Case

input	operation	assertions
1	synchronous	output: 2

Support

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	-	+	-

Table 111: Flow-Links Test

Process name	Flow-Links-JoinCondition
Activities and configuration	assign flow joinCondition links receive reply sequence transitionCondition
Description	A receive-reply pair with an intermediate flow that contains three assigns, two of which point to the third using links. Both links have transitionConditions and their target a joinCondition defined upon them. A joinFailure should result, given not both of the links are activated.
Test case: Good-Case-1	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: joinFailure</td></tr></table>
Test case: Good-Case-2	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>3</td><td>synchronous</td><td>output: 6</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+/-</td><td>+/-</td><td>-</td></tr></table>

Table 112: Flow-Links-JoinCondition Test

Process name	Flow-Links-JoinFailure
Activities and configuration	assign flow joinCondition links receive reply sequence transitionCondition
Description	A receive-reply pair with an intermediate flow that contains three assigns, two of which point to the third using links. Both links have transitionConditions and their target a joinCondition defined upon them. The transitionConditions do never evaluate to true, resulting in a joinFailure on each invocation.
Test case: Good-Case-1	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: joinFailure</td></tr></table>
Test case: Good-Case-2	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>3</td><td>synchronous</td><td>fault: joinFailure</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>-</td><td>-</td><td>-</td></tr></table>

Table 113: Flow-Links-JoinFailure Test

Process name	Flow-Links-ReceiveCreatingInstances
Activities and configuration	assign flow links receive reply sequence
Description	A flow with a starting activity (receive with createInstance set to yes) and a non-starting activity (assign), where a precedence relationship is defined using links.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>output: 6</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table>

Table 114: Flow-Links-ReceiveCreatingInstances Test

<table>
<tr><td>Process name</td><td>Flow-Links-SuppressJoinFailure</td></tr>
<tr><td>Activities and configuration</td><td>assign flow joinCondition links receive reply sequence transitionCondition</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate flow that contains three assigns, two of which point to the third using links. Both links have transitionConditions and their target a joinCondition defined upon them. The transitionConditions do never evaluate to true, resulting in a joinFailure on each invocation. However, this joinFailure is suppressed.</td></tr>
<tr><td>Test case: Good-Case-1</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 3</td></tr></table></td></tr>
<tr><td>Test case: Good-Case-2</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>3</td><td>synchronous</td><td>output: 5</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>-</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 115: Flow-Links-SuppressJoinFailure Test

<table>
<tr><td>Process name</td><td>Flow-Links-TransitionCondition</td></tr>
<tr><td>Activities and configuration</td><td>assign flow links receive reply sequence transitionCondition</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate flow that contains three assigns, two of which point to the third using links. Both links have transitionConditions that do fire only if the input is greater than two.</td></tr>
<tr><td>Test case: Good-Case-1</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 4</td></tr></table></td></tr>
<tr><td>Test case: Good-Case-2</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>3</td><td>synchronous</td><td>output: 6</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+/-</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 116: Flow-Links-TransitionCondition Test

Process name	Flow-ReceiveCreatingInstances
Activities and configuration	assign flow receive reply sequence
Description	A flow with a starting activity (receive with createInstance set to yes) and non-starting activities (assign) that run in parallel. Such a process definition must be rejected by an engine.
Test case: Good-Case	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>5</td><td>synchronous</td><td>NotDeployable</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>+</td><td>-</td></tr></table>

Table 117: Flow-ReceiveCreatingInstances Test

Process name	ForEach
Activities and configuration	assign forEach receive reply scope sequence
Description	A receive-reply pair with an intermediate forEach that loops for n times, where n is equal to the input. Each iteration the current loop number is added to the final result.
Test case: 0-equals-0	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>0</td><td>synchronous</td><td>output: 0</td></tr></table>
Test case: 0plus1-equals-0	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 1</td></tr></table>
Test case: 0plus1plus2-equals-3	<table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 3</td></tr></table>
Support	<table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>+</td></tr></table>

Table 118: ForEach Test

Process name	ForEach-CompletionCondition
Activities and configuration	assign completionCondition forEach receive reply scope sequence
Description	A receive-reply pair with an intermediate forEach that should terminate given two of its children have terminated. N+1 children are scheduled for execution, where n is equal to the input. If N+1 is less than two, an invalidBranchConditionFault should be thrown.
Test case: Skipping the third iteration	input: 2; operation: synchronous; assertions: output: 1
Test case: Cannot meet completion condition	input: 0; operation: synchronous; assertions: fault: invalidBranchCondition
Support	bpel-g: +; ODE: -; openESB: +; Orchestra: -; PetalsESB: -

Table 119: ForEach-CompletionCondition Test

Process name	ForEach-CompletionCondition-NegativeBranches
Activities and configuration	assign completionCondition forEach receive reply scope sequence
Description	A receive-reply pair with an intermediate forEach that should always fail with an invalidExpressionValue fault as branches is initialized with a negative value.
Test case: Iterate-Twice	input: 2; operation: synchronous; assertions: fault: invalidExpressionValue
Support	bpel-g: -; ODE: -; openESB: +; Orchestra: -; PetalsESB: -

Table 120: ForEach-CompletionCondition-NegativeBranches Test

Process name	ForEach-CompletionCondition-Parallel
Activities and configuration	assign completionCondition forEach receive reply scope sequence
Description	A receive-reply pair with an intermediate forEach that should terminate given two of its children have terminated. N+1 children are scheduled for execution in parallel, where n is equal to the input. If N+1 is less than two, an invalidBranchConditionFault should be thrown.
Test case: Skipping the third iteration	input: 2; operation: synchronous; assertions: output: 1
Test case: Cannot meet completion condition	input: 0; operation: synchronous; assertions: fault: invalidBranchCondition
Support	bpel-g: +; ODE: -; openESB: +; Orchestra: -; PetalsESB: -

Table 121: ForEach-CompletionCondition-Parallel Test

Process name	ForEach-CompletionCondition-SuccessfulBranchesOnly
Activities and configuration	assign catch completionCondition empty faultHandlers forEach if receive reply scope sequence throw
Description	A receive-reply pair with an intermediate forEach that should terminate given two of its children have terminated successfully. Each child throws a fault, given the current counter value is even. N children are scheduled for execution, where n is equal to the input.
Test case: Good-Case-1	input: 5; operation: synchronous; assertions: output: 6
Test case: Good-Case-2	input: 10; operation: synchronous; assertions: output: 6
Support	bpel-g: +; ODE: -; openESB: +; Orchestra: -; PetalsESB: -

Table 122: ForEach-CompletionCondition-SuccessfulBranchesOnly Test

Process name	ForEach-CompletionConditionFailure
Activities and configuration	assign catchAll completionCondition empty faultHandlers forEach receive reply scope sequence throw
Description	A receive-reply pair with an intermediate forEach that should terminate given two of its children have terminated. N+1 children are scheduled for execution in parallel, where n is equal to the input. If N+1 is less than two, an invalidBranchConditionFault should be thrown. This is a seperate test case that tests only for the failure.
Test case: Expect completion-ConditionFailure	input: 1; operation: synchronous; assertions: fault: completionConditionFailure
Support	bpel-g: +; ODE: -; openESB: +; Orchestra: -; PetalsESB: -

Table 123: ForEach-CompletionConditionFailure Test

Process name	ForEach-DuplicateCounterVariable
Activities and configuration	assign forEach receive reply scope sequence
Description	A receive-reply pair with an intermediate forEach that contains a variable of the same name as the counter variable in its child scope. The process definition must be rejected.
Test case: SA00076	input: 2; operation: synchronous; assertions: NotDeployable
Support	bpel-g: -; ODE: +; openESB: -; Orchestra: +; PetalsESB: +

Table 124: ForEach-DuplicateCounterVariable Test

<table>
<tr><td>Process name</td><td>ForEach-NegativeStartCounter</td></tr>
<tr><td>Activities and configuration</td><td>assign forEach receive reply scope sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate forEach that should always fail with an invalidExpressionValue fault as startCounterValue is negative.</td></tr>
<tr><td>Test case: Iterate-Twice</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>fault: invalidExpressionValue</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>+</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 125: ForEach-NegativeStartCounter Test

<table>
<tr><td>Process name</td><td>ForEach-NegativeStopCounter</td></tr>
<tr><td>Activities and configuration</td><td>assign forEach receive reply scope sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate forEach that should always fail with an invalidExpressionValue fault as finalCounterValue is negative.</td></tr>
<tr><td>Test case: NegativeStopCounter</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>fault: invalidExpressionValue</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>+</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 126: ForEach-NegativeStopCounter Test

<table>
<tr><td>Process name</td><td>ForEach-Parallel</td></tr>
<tr><td>Activities and configuration</td><td>assign forEach receive reply scope sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate forEach that executes its children in parallel.</td></tr>
<tr><td>Test case: 0plus1plus2-equals-3</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 3</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>-</td><td>+</td></tr></table></td></tr>
</table>

Table 127: ForEach-Parallel Test

<table>
<tr><td>Process name</td><td>ForEach-TooLargeStartCounter</td></tr>
<tr><td>Activities and configuration</td><td>assign forEach receive reply scope sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate forEach that should always fail with an invalidExpressionValue fault as startCounterValue is initialized with a value that exceeds xs:unsignedInt.</td></tr>
<tr><td>Test case: Iterate-Twice</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>fault: invalidExpressionValue</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr></table></td></tr>
</table>

Table 128: ForEach-TooLargeStartCounter Test

<table>
<tr><td>Process name</td><td>If</td></tr>
<tr><td>Activities and configuration</td><td>assign if receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate if that checks whether the input is even.</td></tr>
<tr><td>Test case: Not-If-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 0</td></tr></table></td></tr>
<tr><td>Test case: If-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 129: If Test

<table>
<tr><td>Process name</td><td>If-Else</td></tr>
<tr><td>Activities and configuration</td><td>assign else if receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate if-else that checks whether the input is even.</td></tr>
<tr><td>Test case: Else-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 0</td></tr></table></td></tr>
<tr><td>Test case: If-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 130: If-Else Test

<table>
<tr><td>Process name</td><td>If-ElseIf</td></tr>
<tr><td>Activities and configuration</td><td>assign elseif if receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate if-elseif that checks whether the input is even or divisible by three.</td></tr>
<tr><td>Test case: Not-If-Or-ElseIf-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: 0</td></tr></table></td></tr>
<tr><td>Test case: If-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 1</td></tr></table></td></tr>
<tr><td>Test case: ElseIf-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>3</td><td>synchronous</td><td>output: 2</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>+</td></tr></table></td></tr>
</table>

Table 131: If-ElseIf Test

Process name	If-ElseIf-Else
Activities and configuration	assign else elseif if receive reply sequence
Description	A receive-reply pair with an intermediate if-elseif-else that checks whether the input is even or divisible by three.
Test case: Else-Case	input / operation / assertions: 1 / synchronous / output: 0
Test case: If-Case	input / operation / assertions: 2 / synchronous / output: 1
Test case: ElseIf-Case	input / operation / assertions: 3 / synchronous / output: 2
Support	bpel-g: +, ODE: +, openESB: +, Orchestra: +, PetalsESB: +

Table 132: If-ElseIf-Else Test

Process name	If-InvalidExpressionValue
Activities and configuration	assign if receive reply sequence
Description	A receive-reply pair with an intermediate if that should throw an invalidExpressionValue fault because of an invalid condition.
Test case: SelectionFailure	input / operation / assertions: 1 / synchronous / fault: invalidExpressionValue
Support	bpel-g: +, ODE: -, openESB: -, Orchestra: -, PetalsESB: -

Table 133: If-InvalidExpressionValue Test

Process name	Pick-Correlations-InitAsync
Activities and configuration	assign correlationSets onMessage pick receive reply sequence
Description	An asynchronous receive that initiates a correlationSet, followed by a pick with a synchronous onMessage that correlates on this set.
Test case: Good-Case	input / operation / assertions: 1 / asynchronous / ; 1 / synchronous / output: 1
Support	bpel-g: +, ODE: -, openESB: +, Orchestra: -, PetalsESB: -

Table 134: Pick-Correlations-InitAsync Test

Process name	Pick-Correlations-InitSync
Activities and configuration	assign correlationSets onMessage pick receive reply sequence
Description	A receive-reply pair that initiates a correlationSet, followed by a pick with a synchronous onMessage that correlates on this set.
Test case: Good-Case	input: 1, operation: synchronous, assertions: output: 1 input: 1, operation: synchronous, assertions: output: 2
Support	bpel-g: +, ODE: +, openESB: -, Orchestra: +, PetalsESB: -

Table 135: Pick-Correlations-InitSync Test

Process name	Pick-CreateInstance
Activities and configuration	assign onMessage pick reply sequence
Description	A pick with a synchronous onMessage that has createInstance set to yes.
Test case: Good-Case	input: 1, operation: synchronous, assertions: output: 1
Support	bpel-g: +, ODE: +, openESB: +, Orchestra: +, PetalsESB: +

Table 136: Pick-CreateInstance Test

<table>
<tr><td>Process name</td><td>Pick-OnAlarm-For</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets onAlarm onMessage pick receive reply sequence throw</td></tr>
<tr><td>Description</td><td>An onAlarm with for test case. The test contains a receive-reply pair that initiates a correlationSet and an intermediate pick that contains an onMessage and an onAlarm with an for element. The onAlarm should fire after two seconds and the process should reply with a default value.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: -1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 137: Pick-OnAlarm-For Test

<table>
<tr><td>Process name</td><td>Pick-OnAlarm-Until</td></tr>
<tr><td>Activities and configuration</td><td>assign correlationSets onAlarm onMessage pick receive reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair that initiates a correlationSet and an intermediate pick that contains an onMessage and an onAlarm with an until element. The onAlarm should fire immediately.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>1</td><td>synchronous</td><td>output: -1</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>-</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 138: Pick-OnAlarm-Until Test

<table>
<tr><td>Process name</td><td>RepeatUntil</td></tr>
<tr><td>Activities and configuration</td><td>assign receive repeatUntil reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate while that loops for n+1 times, where n is equal to the input.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 3</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>+</td><td>+</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 139: RepeatUntil Test

<table>
<tr><td>Process name</td><td>RepeatUntilEquality</td></tr>
<tr><td>Activities and configuration</td><td>assign receive repeatUntil reply sequence</td></tr>
<tr><td>Description</td><td>A receive-reply pair with an intermediate while that loops for n times, where n is equal to the input.</td></tr>
<tr><td>Test case: Good-Case</td><td><table><tr><td>input</td><td>operation</td><td>assertions</td></tr><tr><td>2</td><td>synchronous</td><td>output: 2</td></tr></table></td></tr>
<tr><td>Support</td><td><table><tr><td>bpel-g</td><td>ODE</td><td>openESB</td><td>Orchestra</td><td>PetalsESB</td></tr><tr><td>+</td><td>-</td><td>+</td><td>+</td><td>-</td></tr></table></td></tr>
</table>

Table 140: RepeatUntilEquality Test

Process name	Sequence
Activities and configuration	assign receive reply sequence
Description	A receive-reply pair enclosed in a sequence.

Test case: Good-Case

input	operation	assertions
5	synchronous	output: 5

Support

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	+	+	+

Table 141: Sequence Test

Process name	While
Activities and configuration	assign receive reply sequence while
Description	A receive-reply pair with an intermediate while that loops for n times, where n is equal to the input.

Test case: Good-Case

input	operation	assertions
5	synchronous	output: 5

Support

bpel-g	ODE	openESB	Orchestra	PetalsESB
+	+	+	+	+

Table 142: While Test

C Elements Excluded from the Test Descriptions

The following activities, special constructs and attributes are excluded from the *Activities and configuration* row of the test descriptions. The reason for this is that they either appear in every test case or their occurence is implied by the use of other elements that are listed in the row.

- addr:Address
- branches
- condition
- copy
- correlation
- correlationSet
- correlations
- documentation
- ex:foo
- extension
- finalCounterValue
- foo:barEPR
- for
- from
- fromPart
- import
- link
- literal
- messageExchange
- partnerLink
- partnerLinks
- process
- source
- sources
- startCounterValue
- target
- targets
- to
- toPart
- until
- variable
- variables

D List of previous University of Bamberg reports

Bamberger Beiträge zur Wirtschaftsinformatik

Nr. 1 (1989) Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)

Nr. 2 (1990) Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG

Nr. 3 (1990) Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen

Nr. 4 (1990) Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)

Nr. 5 (1990) Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Nr. 6 (1991) Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten

Nr. 7 (1991) Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender

Nr. 8 (1991) Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung

Nr. 9 (1992) Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell

Nr. 10 (1992) Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)

Nr. 11 (1992) Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells

Nr. 12 (1992) Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt

Nr. 13 (1992) Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell

Nr. 14 (1992) Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen

Nr. 15 (1992) Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch

Nr. 16 (1992) Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip

Nr. 17 (1993) Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation

Nr. 18 (1993) Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells

Nr. 19 (1994) Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach

Nr. 20 (1994) Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994

Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -. 2nd edition, November 1994

Nr. 21 (1994) Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen

Nr. 22 (1994) Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems

Nr. 23 (1994) Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle

Nr. 24 (1994) Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen

Nr. 25 (1994) Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme

Nr. 26 (1995) Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes

Nr. 27 (1995) Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995

Nr. 28 (1995) Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach

Nr. 30 (1995) Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit

Nr. 31 (1995) Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse

Nr. 32 (1995) Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger

Nr. 33 (1995) Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?

Nr. 34 (1995) Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -

Nr. 35 (1995) Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme

Nr. 36 (1996) Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996

Nr. 37 (1996) Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996

Nr. 38 (1996) Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996

Nr. 39 (1996) Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze

Nr. 40 (1997) Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997

Nr. 41 (1997) Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik '97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997

Nr. 42 (1997) Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997

Nr. 43 (1997): Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997

Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998

Nr. 44 (1997) Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung

Nr. 45 (1998) Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998

Nr. 46 (1998) Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement". Aachen, September 1997

Nr. 47 (1998) Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt". Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998

Nr. 48 (1998) Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering

Nr. 49 (1998) Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems

Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460

Nr. 50 (1999) Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)

Nr. 51 (1999) Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)

Nr. 52 (1999) Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule" im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999

Nr. 53 (1999) Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999

Nr. 54 (1999) Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.

Nr. 55 (2000) Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur

Nr. 56 (2000) Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000

Nr. 57 (2000) Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.

Nr. 58 (2000) Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.

Nr. 59 (2001) Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001

Nr. 60 (2001) Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001" im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik

Nr. 61 (2002)	Goré R., Mendler M., de Paiva V. (Hrsg.): Proceedings of the International Workshop on Intuitionistic Modal Logic and Applications (IMLA 2002), Copenhagen, July 2002.
Nr. 62 (2002)	Sinz E.J., Plaha M., Ulbrich-vom Ende A.: Datenschutz und Datensicherheit in einem landesweiten Data-Warehouse-System für das Hochschulwesen. Erscheint in: Beiträge zur Hochschulforschung, Heft 4-2002, Bayerisches Staatsinstitut für Hochschulforschung und Hochschulplanung, München 2002
Nr. 63 (2005)	Aguado, J., Mendler, M.: Constructive Semantics for Instantaneous Reactions
Nr. 64 (2005)	Ferstl, O.K.: Lebenslanges Lernen und virtuelle Lehre: globale und lokale Verbesserungspotenziale. Erschienen in: Kerres, Michael; Keil-Slawik, Reinhard (Hrsg.); Hochschulen im digitalen Zeitalter: Innovationspotenziale und Strukturwandel, S. 247 – 263; Reihe education quality forum, herausgegeben durch das Centrum für eCompetence in Hochschulen NRW, Band 2, Münster/New York/München/Berlin: Waxmann 2005
Nr. 65 (2006)	Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet
Nr. 66 (2006)	Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, Karsten Loesing, and Guido Wirtz: Concealing Presence Information in Instant Messaging Systems, April 2006
Nr. 67 (2006)	Marco Fischer, Andreas Grünert, Sebastian Hudert, Stefan König, Kira Lenskaya, Gregor Scheithauer, Sven Kaffille, and Guido Wirtz: Decentralized Reputation Management for Cooperating Software Agents in Open Multi-Agent Systems, April 2006
Nr. 68 (2006)	Michael Mendler, Thomas R. Shiple, Gérard Berry: Constructive Circuits and the Exactness of Ternary Simulation
Nr. 69 (2007)	Sebastian Hudert: A Proposal for a Web Services Agreement Negotiation Protocol Framework . February 2007
Nr. 70 (2007)	Thomas Meins: Integration eines allgemeinen Service-Centers für PC-und Medientechnik an der Universität Bamberg – Analyse und Realisierungs-Szenarien. Februar 2007
Nr. 71 (2007)	Andreas Grünert: Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises. März 2007
Nr. 72 (2007)	Michael Mendler, Gerald Lüttgen: Is Observational Congruence on μ-Expressions Axiomatisable in Equational Horn Logic?
Nr. 73 (2007)	Martin Schissler: out of print
Nr. 74 (2007)	Sven Kaffille, Karsten Loesing: Open chord version 1.0.4 User's Manual. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 74, Bamberg University, October 2007. ISSN 0937-3349.

Nr. 75 (2008) Karsten Loesing (Hrsg.): Extended Abstracts of the Second *Privacy Enhancing Technologies Convention* (PET-CON 2008.1). Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 75, Bamberg University, April 2008. ISSN 0937-3349.

Nr. 76 (2008) Gregor Scheithauer and Guido Wirtz: Applying Business Process Management Systems? A Case Study. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 76, Bamberg University, May 2008. ISSN 0937-3349.

Nr. 77 (2008) Michael Mendler, Stephan Scheele: Towards Constructive Description Logics for Abstraction and Refinement. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 77, Bamberg University, September 2008. ISSN 0937-3349.

Nr. 78 (2008) Gregor Scheithauer and Matthias Winkler: A Service Description Framework for Service Ecosystems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 78, Bamberg University, October 2008. ISSN 0937-3349.

Nr. 79 (2008) Christian Wilms: Improving the Tor Hidden Service Protocol Aiming at Better Performances. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 79, Bamberg University, November 2008. ISSN 0937-3349.

Nr. 80 (2009) Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger, Guido Wirtz: QoS Enabled B2B Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 80, Bamberg University, May 2009. ISSN 0937-3349.

Nr. 81 (2009) Ute Schmid, Emanuel Kitzelmann, Rinus Plasmeijer (Eds.): Proceedings of the ACM SIGPLAN Workshop on *Approaches and Applications of Inductive Programming* (AAIP'09), affiliated with ICFP 2009, Edinburgh, Scotland, September 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 81, Bamberg University, September 2009. ISSN 0937-3349.

Nr. 82 (2009) Ute Schmid, Marco Ragni, Markus Knauff (Eds.): Proceedings of the KI 2009 Workshop *Complex Cognition*, Paderborn, Germany, September 15, 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 82, Bamberg University, October 2009. ISSN 0937-3349.

Nr. 83 (2009) Andreas Schönberger, Christian Wilms and Guido Wirtz: A Requirements Analysis of Business-to-Business Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 83, Bamberg University, December 2009. ISSN 0937-3349.

Nr. 84 (2010) Werner Zirkel and Guido Wirtz: A Process for Identifying Predictive Correlation Patterns in Service Management Systems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 84, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 85 (2010) Jan Tobias Mühlberg und Gerald Lüttgen: Symbolic Object Code Analysis. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 85, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 86 (2010)	Werner Zirkel and Guido Wirtz: Proaktives Problem Management durch Eventkorrelation – ein *Best Practice* Ansatz. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 86, Bamberg University, August 2010. ISSN 0937-3349.
Nr. 87 (2010)	Johannes Schwalb, Andreas Schönberger: Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 87, Bamberg University, September 2010. ISSN 0937-3349.
Nr. 88 (2011)	Jörg Lenhard: A Pattern-based Analysis of WS-BPEL and Windows Workflow. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 88, Bamberg University, March 2011. ISSN 0937-3349.
Nr. 89 (2011)	Andreas Henrich, Christoph Schlieder, Ute Schmid [eds.]: Visibility in Information Spaces and in Geographic Environments – Post-Proceedings of the KI'11 Workshop. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 89, Bamberg University, December 2011. ISSN 0937-3349.
Nr. 90 (2012)	Simon Harrer, Jörg Lenhard: Betsy - A BPEL Engine Test System. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 90, Bamberg University, July 2012. ISSN 0937-3349.