# Zweitveröffentlichung

**Sinz, Elmar J.**

## Konzeptuelle Modellierung der Zustandskonsistenz verteilter betrieblicher Informationssysteme

Datum der Zweitveröffentlichung: 04.03.2024

Verlagsversion (Version of Record), Konferenzveröffentlichung

Persistenter Identifikator: urn:nbn:de:bvb:473-irb-938062

### Erstveröffentlichung

Sinz, Elmar J. (2014): „Konzeptuelle Modellierung der Zustandskonsistenz verteilter betrieblicher Informationssysteme“. In: Hans-Georg Fill, Dimitris Karagiannis, Ulrich Reimer (Hrsg.), Modellierung 2014, Bonn: Gesellschaft für Informatik e. V., S. 241–256, unter: https://dl.gi.de/items/bff2ce47-c5bb-4274-a7bd-4ded21858873.

### Rechtehinweis

# Konzeptuelle Modellierung der Zustandskonsistenz verteilter betrieblicher Informationssysteme

Elmar J. Sinz

Universität Bamberg
Lehrstuhl für Wirtschaftsinformatik,
insbes. Systementwicklung und Datenbankanwendung
Fakultät Wirtschafsinformatik und Angewandte Informatik
96047 Bamberg
elmar.sinz@uni-bamberg.de

**Abstract:** Betriebliche Informationssysteme werden üblicherweise als verteilte Systeme realisiert. Sie bestehen aus mehreren Teilsystemen, jedes von ihnen nimmt während der Ausführung unterschiedliche Zustände an. Damit stellt sich die Frage nach der Zustandskonsistenz des gesamten Systems. Diese kann z. B. verletzt werden, wenn eines der Teilsysteme den von ihm erwarteten Dienst nicht wie geplant erbringt. Der Beitrag geht davon aus, dass die Zustandskonsistenz durch konzeptuelle Modellierung wesentlich unterstützt werden kann, dies aber von den verbreiteten Ansätzen zur Modellierung von Informationssystemen vernachlässigt wird. Am Beispiel des SOM-Ansatzes wird ein Beitrag zur Schließung dieser Forschungslücke vorgeschlagen. Der Anwendungsbezug ergibt sich z. B. im Hinblick auf das verbreitete Paradigma der serviceorientierten Architekturen.

## 1 Einführung

Konzeptuelle Modelle stellen ein wichtiges Hilfsmittel zur Analyse und Gestaltung betrieblicher Informationssysteme (IS), dem informationsverarbeitenden Teilsystem einer Unternehmung, dar. In der Praxis zeichnen sich IS durch eine immer größere Reichweite und Komplexität aus. Gleichzeitig werden sie in hohem Maße als verteilte Systeme realisiert. Dies erfordert wiederum einen hohen Integrationsgrad der Aufgaben und der sie unterstützenden Anwendungssysteme [FeSi13, 237ff].

Konzeptuelle Modelle erfassen Struktur- und Verhaltensmerkmale eines IS aus der fachlichen Sicht der Aufgabenebene. Die Aufgabenträgerebene steht ebenso wie die Frage nach der konkreten Ausgestaltung der Aufgabenträger bei der konzeptuellen Modellierung nicht im Vordergrund. Die Unterscheidung zwischen der Aufgaben- und der Aufgabenträgerebene macht Freiheitsgrade bezüglich der Gestaltung eines IS, z. B. hinsichtlich unterschiedlicher Automatisierungsgrade und -formen, sichtbar und unterstützt die laufende Abstimmung zwischen „Business“ und „IT“. Ganzheitliche konzeptuelle Modelle bilden somit eine wichtige Grundlage für ein von allen Beteiligten getragenes

Fachverständnis der Aufgabenebene des IS sowie für die Identifikation von Gestaltungsoptionen auf der Aufgaben- und Aufgabenträgerebene.

Typischerweise werden bei der konzeptuellen Modellierung sowohl struktur- als auch verhaltensorientierte Sichten auf das IS erfasst. Strukturorientierte Sichten werden insbesondere in Form von Datenschemata (z. B. im Entity-Relationship-Modell ERM [Chen76] oder im Strukturierten Entity-Relationship-Modell SERM [Sinz88]) oder in Form von objektorientierten Klassenschemata (z. B. UML-Klassendiagramme [UML11]) modelliert. Verhaltensorientierte Sichten beziehen sich auf den Ablauf von Aufgaben (je nach Modellierungssprache auch Funktionen, Aktionen, Prozessschritte oder Aktivitäten). Sie werden z. B. in Form von Ereignisgesteuerten Prozessketten [Nütt13], UML-Aktivitätsdiagrammen, UML-Sequenzdiagrammen oder BPMN-Diagrammen [BPMN11] spezifiziert.

Bemerkenswert ist, dass die Sicht auf die Zustände eines IS und auf Konsistenzbedingungen für diese Zustände im Rahmen der konzeptuellen Modellierung kaum Beachtung findet. Dies ist insofern verwunderlich, als IS aufgrund ihrer Verteiltheit (zu verteilten Systemen siehe [Ens78]) aus einer Vielzahl von interagierenden Teilsystemen und Systemkomponenten bestehen, deren lokale Zustände aus globaler Systemsicht konsistent zu halten sind. Die relevanten Zustände eines IS und die zugehörigen Konsistenzbedingungen betreffen somit fachliche Fragen, die auf der Aufgabenebene zu beantworten sind, wofür sich konzeptuelle Modelle als unterstützendes Hilfsmittel anbieten. Konzeptuelle Zustandsmodelle liefern darüber hinaus wichtige Anforderungen für die Gestaltung der Zustandskonsistenz auf der Aufgabenträgerebene, z. B. mithilfe von (verteilten) Datenbanktransaktionen und Transaktionsdiensten.

Der vorliegende Beitrag geht von der Arbeitshypothese aus, dass konzeptuelle Modelle von Zuständen und zugehörigen Konsistenzbedingungen einen wichtigen Beitrag zur ganzheitlichen fachlichen Analyse und Gestaltung verteilter IS und damit zur Beherrschung ihrer Komplexität leisten können. Konzeptuelle Zustandsmodelle treten dabei als „dritte Sicht“ neben die strukturorientierte Sicht und die Ablaufsicht auf IS. Ebenso wie die Ablaufsicht zielt auch die Zustandssicht auf Verhaltenseigenschaften eines IS.

Ein Beispiel ist das verteilte System *Reisebuchungssystem*, bestehend aus den Teilsystemen *Agentur*, *Flugline*, *Autovermietung* und *Hotel*. Unterstellt man, dass nur eine vollständige Reise akzeptiert wird und dass die Buchung in der Reihenfolge Flug, Mietwagen und Hotel erfolgt, so sind aus konzeptueller Sicht die Schritte zu beschreiben, die auf *Agentur*, *Fluglinie* und *Autovermietung* durchzuführen sind, wenn in der Zielstadt das gewünschte Hotel nicht verfügbar ist und auch kein alternatives Hotel gefunden werden kann. Auch kann die Zustandssicht Hinweise für eine Umgestaltung des Geschäftsprozesses geben, indem z. B. der Erwartungswert für die Durchführbarkeit einer gewünschten Buchung oder die Stornierungskosten und –fristen für eine Fehlbuchung berücksichtigt werden.

Im Folgenden wird ein Ansatz zur konzeptuellen Modellierung der Zustandskonsistenz von IS, d. h. der Modellierung relevanter Zustände und zugehöriger Konsistenzbedingungen, vorgeschlagen. Ziel ist es, bereits auf konzeptueller Modellebene Aussagen über

die Konsistenz von Systemzuständen eines verteilten IS treffen und Anforderungen zu deren Erreichung gezielt erfassen zu können. Hierzu gehören die Sichtbarmachung konsistenter und nicht-konsistenter Systemzustände sowie die Identifikation von Maßnahmen zur Erreichung konsistenter Zustände. Als Forschungsansatz dient eine deduktive Analyse auf der Grundlage von Systemtheorie und Kybernetik. Der Ansatz wird anhand der SOM-Methodik ([FeSi90, [FeSi91), [FeSi95], [FeSi13]) ausgeführt.

Der weitere Beitrag gliedert sich wie folgt: In Abschnitt 2 werden Grundlagen der Modellierung und der Konsistenzsicherung von Systemzuständen vorgestellt. Der Ansatz zur Modellierung der Zustandskonsistenz betrieblicher Informationssysteme wird in Abschnitt 3 entwickelt. Abschnitt 4 führt den Ansatz anhand der Fallstudie „Reisebuchung“ durch. Abschnitt 5 widmet sich der Diskussion des vorgestellten Ansatzes.

# 2 Grundlagen der Modellierung und Konsistenzsicherung von Zuständen

## 2.1 Modellierung von Systemzuständen

Systemtheoretische Grundlagen der Modellierung von Zuständen finden sich insbesondere im Systemtyp *endlicher Automat*. Ein endlicher Automat kann eine endliche Menge von Zuständen annehmen. Durch einen Stimulus (Eingabe) geht das System von einem Vorzustand in einen Nachzustand über und erzeugt eine Reaktion (Ausgabe). Handelt es sich um einen endlichen Automaten mit einer diskreten Zustandsmenge und einer überschaubaren Anzahl von Zuständen, so kann das Verhalten des Automaten in Form eines Zustandsüberführungsgraphen (Systemtyp *Zustandsraum-System*) beschrieben werden [FeSi13, 13ff].

Ein weiterer Systemtyp ist das *Petri-Netz*. Ein Petri-Netz umfasst eine Menge von Zuständen und Übergängen, welche die Knoten des Petri-Netzes bilden und die durch gerichtete Kanten zu einem bipartiten Graphen verbunden sind. Den Zuständen können Marken (*Token*) zugeordnet werden. Durch Schalten von Übergängen werden Marken aus den Vorzuständen eines Übergangs entfernt und Nachzustände markiert. Im Gegensatz zu einem Zustandsüberführungsgraphen, bei dem der aktuelle Systemzustand in jeweils genau einem Knoten lokalisiert werden kann, wird der Systemzustand eines Petri-Netzes durch seine aktuelle (in der Regel mehrelementige) Menge von Marken und deren Verteilung auf die Zustandsknoten definiert [Reis10].

Ausdrucksmittel zur Modellierung von Zuständen auf Basis der beiden Systemtypen finden sich in einer Reihe von Modellierungssprachen. Angesichts der weiten Verbreitung der Unified Modeling Language (UML) wird exemplarisch der UML-Zustandsautomat (*state machine*) kurz angesprochen. Diese Diagrammart erlaubt eine differenzierte Modellierung von Zuständen und Übergängen eines Teilsystems (*behavioral state machine*) sowie eine Modellierung des Nutzungsprotokolls von Systemschnittstellen (*protocol state machine*) [UML11, 535ff]. Der zugrunde liegende Systemtyp ist der endliche Automat. Zustände können aus einfacheren Zuständen zusammengesetzt sein. Teilzustände zusammengesetzter Zustände lassen sich Regionen (*regions*) zuord-

nen. Damit können Teil-Zustandsautomaten mit nebenläufigen Zustandsübergängen gebildet werden. Das Gesamtsystem wird damit als Petri-Netz darstellbar.

Eine konzeptuelle, semantische Modellierung von Systemzuständen auf der Aufgabenebene von IS ist in Wissenschaft und Praxis jedoch nur wenig verbreitet. Zustandsmodelle werden überwiegend auf der Aufgabenträgerebene und hier meist nur für kleinere Teil-Anwendungssysteme oder IT-Infrastrukturkomponenten genutzt. Dadurch bleiben wichtige Analyse- und Gestaltungspotenziale für IS ungenutzt.

## 2.2 Transaktionskonzept und Transaktionsmodelle

Das aus dem Bereich der Datenbanksysteme bekannte Transaktionskonzept definiert eine Transaktion als eine Folge von Operationen auf einer Datenbasis, welche einen konsistenten Zustand der Datenbasis in einen wiederum konsistenten Zustand überführt [Reut87, 405]. Die Eigenschaften von Transaktionen werden im ACID-Prinzip zusammengefasst: Eine Transaktion ist nicht unterbrechbar, sie wird nach dem Alles-oder-nichts-Prinzip durchgeführt (*Atomicity*), sie wahrt die Konsistenz der Datenbasis (*Consistency*), mehrere Instanzen von Transaktionen laufen ohne gegenseitige Beeinflussung ab (*Isolation*) und die durch eine Transaktion bewirkten Veränderungen der Datenbasis sind dauerhaft (*Durability*).

Die Operationen einer Transaktion sind durch eine sogenannte Kontrollsphäre geschützt [GrRe93, 174]. Eine Kontrollsphäre lässt sich als Instanz eines abstrakten Datentyps interpretieren. Der Aufruf einer Operation an der Schnittstelle des abstrakten Datentyps führt zu einer Operationsfolge in dessen Innerem, die nach dem Alles-oder-nichts-Prinzip ausgeführt wird. Erst nach vollständiger Durchführung der Operationsfolge wird über die Schnittstelle ein Ergebnis nach außen bekannt gemacht.

Jedes transaktionsgeschützte System ist aus Sicht seiner Kontrollsphären in Form von Hierarchien abstrakter Datentypen darstellbar. Bei einfachen (flachen) Transaktionen besteht die Hierarchie nur aus dem Wurzelknoten. Bei höheren Transaktionsmodellen, wie genesteten Transaktionen und Mehrebenen-Transaktionen [GrRe93, 195ff] sind die Hierarchien als Baumstruktur ausgeprägt.

Bei genesteten Transaktionen sind die Kontrollsphären der abstrakten Datentypen verschachtelt, d. h. jeder Wurzelknoten umschließt die ihm nachgeordneten Knoten. Das bedeutet, dass kein Blattknoten oder Wurzelknoten eines Teilbaums isoliert in den Zustand *dauerhaft* übergehen kann (*Commitment*), sondern nur gemeinsam mit dem Wurzelknoten des gesamten Baumes. Bis zum Ende dieser Wurzeltransaktion kann somit das gesamte System von Transaktionen zurückgesetzt werden.

Bei Mehrebenen-Transaktionen sind die Kontrollsphären der abstrakten Datentypen nicht verschachtelt. Jede Transaktion eines Blattknotens kann isoliert in den Zustand dauerhaft übergehen, das gleiche gilt für die Transaktion einer Teilbaum-Wurzel, wenn alle nachgelagerten Transaktionen abgeschlossen sind. Da dauerhaft gewordene Transaktionen nicht mehr zurückgesetzt werden können, erfordert das Modell der Mehrebenen-Transaktionen die Installation von kompensierenden Transaktionen, welche im Fehlerfall die Wirkung einer bereits dauerhaft abgeschlossenen Transaktion aufheben.

Wichtig ist, dass die Konsistenz des gesamten Systems aus konzeptueller Sicht durch eine Menge von ACID-Transaktionen definiert wird. Erweiterungen des ACID-Prinzips, z. B. Brewer's CAP-Theorem [Tiwa11, 174f], wonach ein System stets nur zwei der drei Eigenschaften *Consistency*, *Availability* und *Partion Tolerance* erfüllen kann, sind keine Frage der Gestaltung der Aufgabenebene, sondern betreffen die Aufgabenträgerebene.

### 2.3 IT-Unterstützung der Zustandskonsistenz

Hilfsmittel zur Unterstützung von Zustandskonsistenz liegen in höheren Programmiersprachen in Form von Sprachelementen zur Ausnahmebehandlung vor (z. B. in Java der *try-catch*-Block). Tritt bei der Ausführung des *try*-Blocks ein Ausnahmeereignis ein, so wird ein nachfolgender *catch*-Block ausgeführt, der dieses Ausnahmeereignis behandelt.

Diese rudimentäre Form der Ausnahmebehandlung weist keinerlei Bezug zum Transaktionskonzept auf. Zur Unterstützung der Ausführung von Transaktionen in Anwendungssystemen werden spezielle Funktionskomponenten eingesetzt, die als Transaktionsmanager [GrRe93, 21] bezeichnet werden. Transaktionsmanager werden entweder von der Systemsoftware (z. B. Datenbankverwaltungssysteme) oder von der Middleware in Form von Transaktionsdiensten (z. B. Java Transaction Service JTS) bereitgestellt. JTS stellt Transaktionsdienste speziell für verteilte Anwendungssysteme bereit. Standardmäßig werden flache Transaktionen, optional auch genestete Transaktionen unterstützt.

Workflow-Sprachen wie WS-BPEL gehen davon aus, dass der Einsatz von ACID-Transaktionen mit einem gemeinsamen Commitment am Ende der Wurzeltransaktion für Workflows nur bedingt möglich ist, da die zugehörigen Prozessinstanzen eine längere Laufzeit aufweisen können, so dass die Transaktionseigenschaft der Isolation nicht oder nur eingeschränkt garantiert werden kann [OAS07, 117ff]. WS-BPEL weicht daher auf das Konzept der Mehrebenen-Transaktion aus und unterstützt das Prinzip der Kompensation durch die Konzepte *Scope* und *Compensation Handler*.

## 3 Ein Ansatz zur Modellierung der Zustandskonsistenz betrieblicher Informationssysteme

Im Folgenden wird der im Mittelpunkt der Arbeit stehende Ansatz zur Modellierung von Systemzuständen und deren Konsistenzbedingungen vorgestellt und diskutiert. Die Entwicklung des Ansatzes erfolgt auf der Grundlage von Geschäftsprozessmodellen gemäß der SOM-Methodik ([FeSi90], (FeSi91], [FeSi95], [FeSi13, 194ff]). Die SOM-Methodik ist wegen ihrer strikten Objektorientierung in besonderer Weise für eine konzeptuelle Zustandsmodellierung geeignet. Kern dieser Objektorientierung sind gekapselte betriebliche Objekte, die lose gekoppelt sind und mithilfe von betrieblichen Transaktionen koordiniert werden. In Bezug auf die Zustandsmodellierung sind folgende Eigenschaften bedeutsam: Objekte besitzen jeweils ihren eigenen Objektspeicher (ihr „fachliches Gedächtnis") und kapseln ihre Zustände. Es gibt keine gemeinsamen Zustände mehrerer Objekte aufgrund eines gemeinsamen Objektspeichers. Eine betriebliche Transaktion zwischen zwei Objekten führt zu abgestimmten Zustandsübergängen der an der Transaktion beteiligten Objekte. Jeder Zustandsübergang eines Objekts stellt eine ACID-

Transaktion dar. Durch die synchrone Kopplung der beiden Zustandsübergänge zweier Objekte in einer betrieblichen Transaktion werden diese zu einer ACID-Transaktion vereinigt.

Der Modellierungsansatz wird zunächst in Abschnitt 3.1 anhand eines einfachen Beispiels entwickelt. Anschließend erfolgt in Abschnitt 3.2 die nähere methodische Fundierung.

### 3.1 Konzeptuelle Modellierung der Zustandskonsistenz auf Basis der SOM-Methodik

Abbildung 1 zeigt in der linken und in der mittleren Spalte das Interaktionsschema (Strukturorientierte Leistungs- und Lenkungssicht) und das Vorgangs-Ereignis-Schema (verhaltensorientierte Ablaufsicht) für ein einfaches Geschäftsprozessmodell gemäß SOM-Methodik. Beide Sichten sind auf einer aggregierten (obere Zeile) und einer detaillierten (untere Zeile) Zerlegungsebene dargestellt. In der rechten Spalte kommt als „dritte Sicht" das konzeptuelle Modell der Zustände und ihrer Konsistenzbedingungen (verhaltensorientierte Zustandssicht) in Form eines Zustandsschemas hinzu. Auch dieses wird auf den beiden genannten Zerlegungsebenen dargestellt.

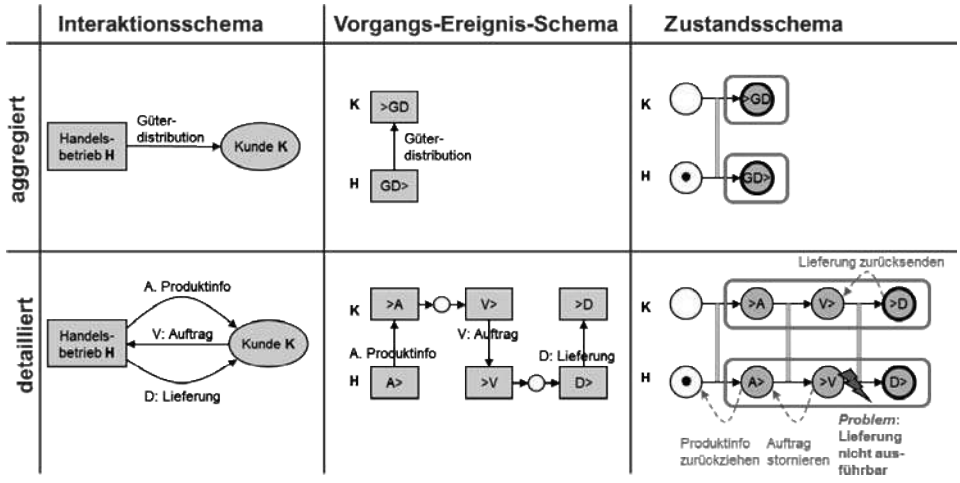


**Abbildung 1:** Interaktionsschema und Vorgangs-Ereignis-Schema gemäß SOM-Methodik, ergänzt um Zustandsschema

Das Interaktionsschema der aggregierten Ebene zeigt die Leistungs- und Lenkungssicht des Geschäftsprozessmodells. Diese umfasst eine Leistungstransaktion *Güterdistribution*, die von dem betrieblichen Diskursweltobjekt *Handelsbetrieb* zu dem betrieblichen Umweltobjekt *Kunde* führt. Auf der detaillierten Ebene wird diese Leistungstransaktion nach dem Verhandlungsprinzip in eine Anbahnungstransaktion *Produktinfo*, eine Vereinbarungstransaktion *Auftrag* und eine Durchführungstransaktion *Lieferung* zerlegt.

Neben den Interaktionsschemata der beiden Zerlegungsebenen sind die korrespondierenden Vorgangs-Ereignis-Schemata dargestellt. Auf der aggregierten Ebene besteht der Ablauf des Geschäftsprozesses im Versenden eines Leistungspaketes *Güterdistribution* durch die Aufgabe *GD>* des Objekts *Handelsbetrieb* sowie im Entgegennehmen dieses Leistungspakets durch die Aufgabe *>GD* des Objekts *Kunde*. Auf der detaillierten Ebene werden die drei Teiltransaktionen *Produktinfo*, *Auftrag* und *Lieferung* sequenziell durchgeführt.

Jede Durchführung eines Geschäftsprozesses wird durch ein initiales Ereignis (Geschäftsvorfall) ausgelöst. Im detaillierten Vorgangs-Ereignis-Schema in Abbildung 1 besteht das initiale Ereignis im Auslösen einer Durchführung der Aufgabe *A>* des Objekts *Handelsbetrieb*. Die Aufgabe *A>* führt gemeinsam mit der Aufgabe *>A* von *Kunde* die Transaktion *Produktinfo* durch. Wichtig ist, dass beide Aufgaben nur gemeinsam die Transaktion ausführen können, d. h. die Transaktion ist nur dann erfolgreich abgeschlossen, wenn beide Aufgaben vollständig durchgeführt wurden.

An dieser Stelle werden die Zusammenhänge zwischen betrieblichen Transaktionen und konsistenzerhaltenden Zustandsübergängen betrieblicher Objekte sichtbar: Die betriebliche Transaktion *Produktinfo* wird gemeinsam von den beiden Aufgaben *A>* und *>A* durchgeführt. Die beiden Aufgaben operieren auf dem Speicher der Objekte *Handelsbetrieb* bzw. *Kunde*. Jede der beiden Aufgabendurchführungen stellt eine ACID-Transaktion (siehe Abschnitt 2.2) dar und führt zu einem konsistenzerhaltenden Zustandsübergang im jeweiligen Objekt. Durch eine Aufgabendurchführung geht der Speicher des jeweiligen betrieblichen Objekts von einem konsistenten Vorzustand in einen wiederum konsistenten Nachzustand über. Anhand der betrieblichen Transaktion *Produktinfo* sind die beiden Aufgabendurchführungen gekoppelt, d. h. die beiden ACID-Transaktionen können nur gemeinsam durchgeführt werden. Mit anderen Worten, ihre beiden Kontrollsphären sind durch eine umfassende Kontrollsphäre miteinander verbunden. In analoger Weise erfolgen gekoppelte Aufgabendurchführungen für die Transaktionen *Auftrag* und *Lieferung*.

In der „dritten Sicht“, dem Zustandsschema, steht nun die Entwicklung der Zustände der an einem Geschäftprozess beteiligten Objekte im Vordergrund. Im Zustandsschema der aggregierten Ebene gehen die Objekte *Handelsbetrieb* und *Kunde* bei der Durchführung der Aufgaben *GD>* bzw. *>GD* von einem Startzustand in einen konsistenten Folgezustand über. Diese Nachzustände der Aufgabendurchführungen werden mit dem Namen der Aufgabe bezeichnet, d. h. der Zustand *GD>* ist der Nachzustand der Durchführung der Aufgabe *GD>*. Da die beiden Aufgabendurchführungen *GD>* und *>GD* durch die betriebliche Transaktion *Güterdistribution* gekoppelt sind, gehen die Objekte *Handelsbetrieb* und *Kunde* synchron in die Zustände *GD>* bzw. *>GD* über.

Auf der aggregierten Ebene führt die Durchführung des Geschäftsprozesses in den Objekten *Handelsbetrieb* und *Kunde* lediglich zu jeweils einem Zustandsübergang und folglich einem zugehörigen Folgezustand, welcher gleichzeitig den Endzustand des jeweiligen Objekts im Rahmen der jeweiligen Geschäftsprozessdurchführung darstellt. Auf der detaillierten Ebene besteht die Durchführung des Geschäftsprozesses aus drei Teiltransaktionen, welche in drei Aufgabenpaaren der Objekte *Handelsbetrieb* und *Kun-*

*de* ausgeführt werden. Auf den Startzustand der Objekte folgt somit eine Sequenz von drei Nachzuständen, die aufgrund der gekoppelten Aufgabendurchführungen (Zustandsübergänge) synchron erreicht werden.

Jeder Nachzustand einer transaktionsbezogenen Aufgabendurchführung stellt einen konsistenten Zwischen- oder Endzustand des jeweiligen betrieblichen Objekts in Bezug auf die aktuelle Instanz einer Geschäftsprozessdurchführung dar. Ein konsistenter Endzustand des gesamten betrieblichen Systems in Bezug auf die aktuelle Geschäftsprozessinstanz wird dann erreicht, wenn alle an der Geschäftsprozessdurchführung beteiligten betrieblichen Objekte einen Endzustand erreicht haben. Ist dies nicht der Fall, d h. es befinden sich zwei oder mehrere betriebliche Objekte nicht in einem Endzustand, so liegt bezüglich der Ausführung der jeweiligen Geschäftsprozessinstanz ein inkonsistenter Zustand vor. Die Ausführung des Geschäftsprozesses muss in diesem Fall fortgesetzt, oder wenn dies nicht möglich ist, rückabgewickelt werden.

Zum Beispiel könnte bei dem Beispiel in Abbildung 1 die Situation eintreten, dass die Lieferung der beauftragten Güter nicht durchgeführt werden kann. In diesem Fall kann der gekoppelte Zustandsübergang von *>V* nach *D>* in *Handelsbetrieb* und von *V>* nach *>D* in *Kunde* nicht vollzogen werden. Der Ausführungszustand des betrieblichen Systems ist in Bezug auf die vorliegende Geschäftsprozessinstanz inkonsistent, da zwar ein gültiger Auftrag vorliegt, dieser aber nicht ausgeführt werden kann. In diesem Fall ist es notwendig, die Zwischenzustände der Objekte sukzessive in Startzustände zurückzuführen und damit die Wirkungen der durchgeführten betrieblichen Transaktionen bzw. ihrer zugehörigen ACID-Transaktionen zu kompensieren.

Um dies zu ermöglichen ist im konzeptuellen Zustandsschema bei *Handelsbetrieb* ein kompensierender Zustandsübergang *Auftrag stornieren* von *>V* nach *A>* vorzusehen, der mit einem korrespondierenden Zustandsübergang bei Kunde von *V>* nach *>A* gekoppelt ist. Weiter ist in *Handelsbetrieb* ein Zustandsübergang *Produktinfo zurückziehen* von *A>* zum Startzustand und gekoppelt damit bei *Kunde* ein Übergang von *>A* zum Startzustand vorzusehen, welcher z. B. die übersandten Produktinformationen als ungültig erklärt. Nach der Durchführung dieser beiden Zustandsübergänge sind die Wirkungen der unvollständig ausgeführten Geschäftsprozessinstanz kompensiert und das betriebliche System befindet sich wiederum in einem konsistenten Zustand.

Die kompensierenden Zustandsübergänge lassen sich als inverse Transaktionen (*Produktinfo zurückziehen* als inverse Transaktion zu *Produktinfo*) interpretieren. Diese inversen Transaktionen müssen allerdings mit semantischem Bezug zum jeweiligen Geschäftsprozessmodell spezifiziert werden. Sie sind somit nicht einfach technischer Natur, sondern Gegenstand der konzeptuellen Modellierung. Zum Beispiel kann die zu *Produktinfo* inverse Transaktion im einen Fall darin bestehen, dass die Produktinformationen gegenüber dem Kunden als ungültig erklärt werden, im anderen Fall in einer leeren Transaktion, welche die übersandten Produktinformationen unverändert beim Kunden belässt.

### 3.2 Methodische Fundierung und Ausführungsmodell von Zustandsschemata

Die in Abschnitt 3.1 verwendeten Modellbausteine zur Darstellung von Zustandsschemata sind in Abbildung 2 in Form einer Legende zusammengefasst. Der Zustandsraum

eines betrieblichen Objekts wird durch Umrandung seiner konsistenten Zwischen- und Endzustände gekennzeichnet. Die Beschreibung der Zustände folgt dem Automatenkonzept, d. h. ein betriebliches Objekt ist zu jedem Zeitpunkt durch genau einen Zustand beschrieben.

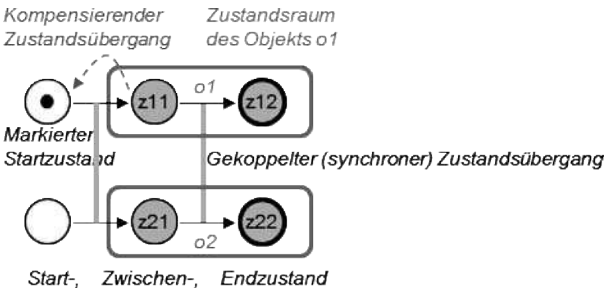


**Abbildung 2:** Legende zu den Modellbausteinen für Zustandsschemata

Jede Durchführung einer Instanz eines Geschäftsprozesses wird durch ein auslösendes Ereignis initiiert, welches mit einem Geschäftsvorfall korrespondiert. Geschäftsvorfälle können von Objekten der Umwelt (z. B. Kundenauftrag) oder von Objekten der Diskurswelt (z. B. Bestellanforderung) ausgelöst werden. Startzustände, die mit Geschäftsvorfällen korrespondieren, sind speziell markiert.

Jedes an der Durchführung einer Geschäftsprozessinstanz beteiligte betriebliche Objekt durchläuft ausgehend von einem Startzustand eine Folge von Zwischenzuständen bis zu einem Endzustand. Die Geschäftsprozessinstanz ist vollständig ausgeführt, wenn alle beteiligten Objekte einen Endzustand erreicht haben. Von einem Startzustand eines betrieblichen Objekts kann es mehrere Wege geben, die zu unterschiedlichen Endzuständen führen. Die von einem gegebenen Zustand erreichbaren Folgezustände generieren somit eine Baumstruktur (in den hier gezeigten Beispielen degeneriert die Baumstruktur zu einer linearen Liste).

Die Bezeichner der Zwischen- und Endzustände werden aus dem Namen der jeweiligen Aufgabe gebildet. Diese Konvention unterstreicht, dass sich das Sachziel einer Aufgabe auf den Nachzustand ihrer Durchführung bezieht. Zudem lässt sich auf diese Weise die Beziehung zwischen Vorgangs-Ereignis-Schema und Zustandsschema leicht nachvollziehen.

In Bezug auf die Durchführung einer Geschäftsprozessinstanz werden die in Abbildung 3 zusammengefassten Objektbereiche und zugehörige Konsistenzbedingungen unterschieden.

| Objektbereich | Konsistenzbedingung |
|---|---|
| Aufgabe | Die Durchführung einer Aufgabe besteht in der Ausführung eines Lösungsverfahrens auf einem Aufgabenobjekt. Das Lösungsverfahren wird in einer ACID-Transaktion gekapselt. Die vollständige und korrekte Durchführung einer Aufgabe führt zu einem konsistenten Nachzustand ihres Aufgabenobjekts. |
| Betriebliche Transaktion | Eine betriebliche Transaktion wird von zwei Aufgaben durchgeführt, die einen gekoppelten Zustandsübergang von zwei Aufgabenobjekten realisieren. Die beiden aufgabenspezifischen ACID-Transaktionen sind zu einer umfassenden ACID-Transaktion gekoppelt. |
| Betriebliches Objekt | Ein betriebliches Objekt umfasst die Aufgabenobjekte der zugehörigen Aufgaben. Die aus der Sicht der einzelnen Aufgaben konsistenten Nachzustände stellen aus der Sicht des betrieblichen Objekts nicht-konsistente Zwischenzustände dar. Erst mit dem Erreichen eines Endzustands ist ein konsistenter Zustand des gesamten betrieblichen Objekts erreicht. |
| Betriebliches System | Ein betriebliches System umfasst eine Menge betrieblicher Objekte. Dieses befindet sich in einem konsistenten Zustand, wenn alle an der Durchführung einer Geschäftsprozessinstanz beteiligten betrieblichen Objekte einen Endzustand erreicht haben. |

**Abbildung 3:** Konsistenzebenen und zugehörige Konsistenzbedingungen

Wird bei der Durchführung einer Geschäftsprozessinstanz ein konsistenter Zustand des betrieblichen Systems nicht erreicht, so sind die Zustände der beteiligten betrieblichen Objekte durch kompensierende Zustandsübergänge in Startzustände zurückzuführen. Die kompensierenden Zustandsübergänge entsprechen inversen Transaktionen, die mit semantischem Bezug zum jeweiligen Geschäftsprozess spezifiziert werden.

Das Konzept des kompensierenden Zustandsübergangs unterstellt, dass im Gegensatz zum Rücksetzen einer Transaktion die Information aus der Transaktion im „Gedächtnis" der betrieblichen Objekte erhalten bleibt und lediglich ihre fachliche Wirkung kompensiert wird. Ein bekanntes Beispiel aus der kaufmännischen Buchführung verdeutlicht dies: Die Durchführung einer Buchung im System der doppelten Buchführung umfasst einen Eintrag im Sollkonto, einen Eintrag im Habenkonto und einen Journaleintrag. Die gesamte Buchung ist zu einer ACID-Transaktion gekapselt. Bricht die Transaktion z. B. nach der Habenbuchung, aber vor dem Journaleintrag ab, so wird sie zurückgesetzt ohne dabei Spuren zu hinterlassen. Wurde die Buchung aber abgeschlossen (die Transaktion ist dauerhaft) und stellt sich nachträglich als sachlich falsch heraus, so muss die fachliche Wirkung durch eine Gegenbuchung kompensiert werden. Die Informationen aus der

ursprünglichen Buchung und der Gegenbuchung bleiben jedoch in den Konten und im Journal dauerhaft erhalten.

## 4 Fallbeispiel: Reisebuchung

Ein Kunde beauftragt eine Agentur (Reisebüro) mit der Buchung einer Reise. Die zu buchenden Leistungen umfassen Flug, Mietwagen und Hotel. Nur wenn alle drei Bestandteile gebucht werden können, ist das Sachziel der Reisebuchung erreicht. Das Interaktionsschema des Geschäftsprozesses ist in Abbildung 4 dargestellt. Es umfasst die betrieblichen Objekte *Kunde* (Umweltobjekt) sowie *Agentur*, *Fluglinie*, *Autovermietung* und *Hotel* (Diskursweltobjekte). Beauftragungen werden durch Vereinbarungstransaktionen (V), die zugehörige Leistungserbringung in Form von Durchführungstransaktionen (D) modelliert. Die lose gekoppelten und mithilfe von betrieblichen Transaktionen koordinierten Objekte stellen ein verteiltes System dar.

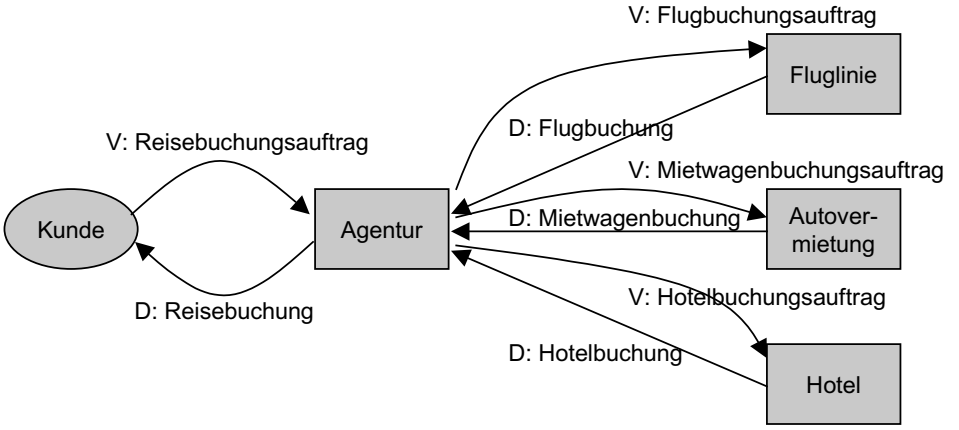


**Abbildung 4:** Interaktionsschema (IAS) für das Beispiel Reisebuchung

Das zugehörige Vorgangs-Ereignis-Schema zeigt Abbildung 5. Die Bezeichner der Aufgaben der betrieblichen Objekte sind aus den Namen der Transaktionen abgeleitet, welche diese Aufgaben durchführen. Zum Beispiel wird die Transaktion *V: Reisebuchungsauftrag* durch die Aufgabe *RA>* („Senden Reisebuchungsauftrag“) von *Kunde* und *>RA* („Empfangen Reisebuchungsauftrag“) von *Agentur* durchgeführt. Aufgaben innerhalb eines Objekts werden durch objektinterne Ereignisse verknüpft (z. B. *>FB* und *MA>* im Objekt *Agentur*). Auf diese Weise werden Reihenfolgebeziehungen zwischen Transaktionen hergestellt. Jedes objektinterne Ereignis korrespondiert mit dem Nachzustand einer vorausgehenden Aufgabendurchführung und dem Vorzustand der folgenden Aufgabendurchführung (im genannten Beispiel korrespondiert das objektinterne Ereignis mit dem Nachzustand der Aufgabe *>FB* und dem Vorzustand der Aufgabe *MA>*).

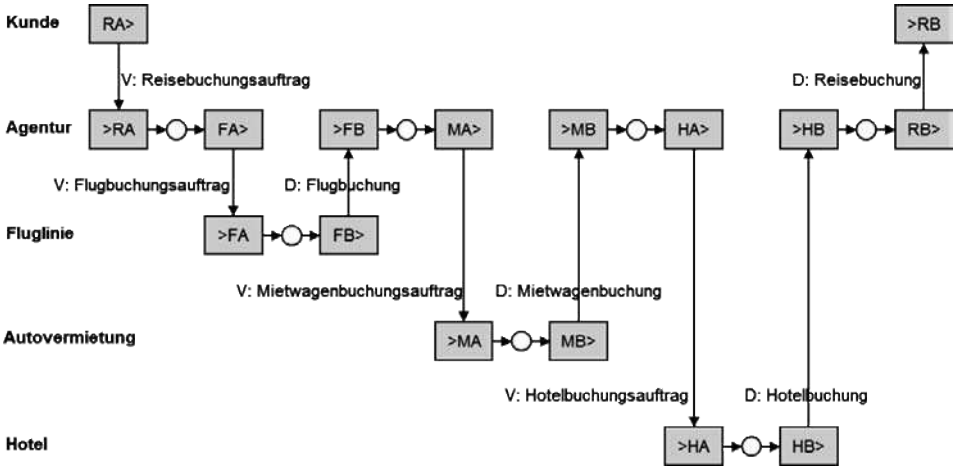


**Abbildung 5:** Vorgangs-Ereignis-Schema (VES) für das Beispiel Reisebuchung

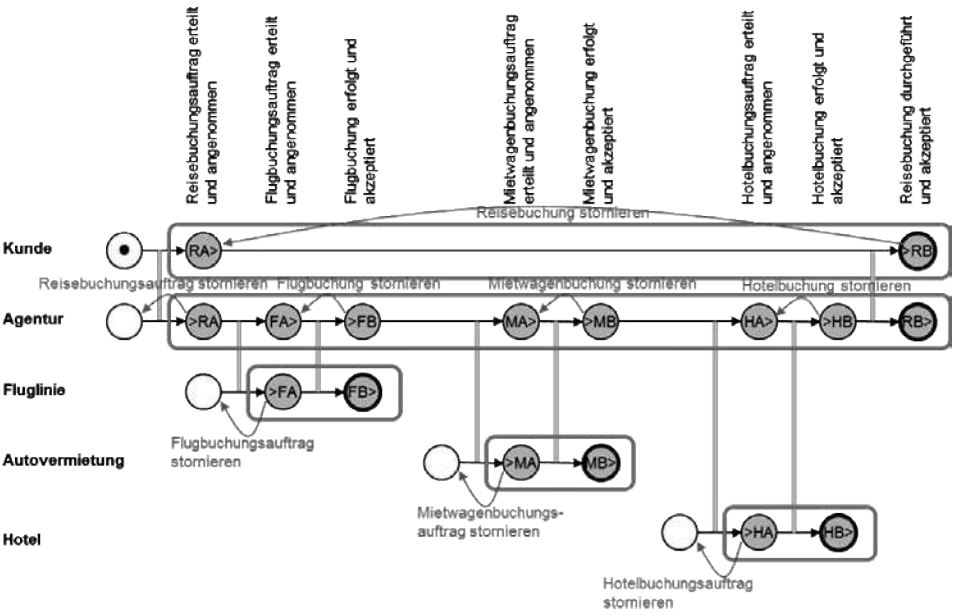


**Abbildung 6:** Zustandsschema für das Beispiel Reisebuchung

Abbildung 6 zeigt das Zustandsschema als „dritte Sicht“ für das Beispiel Reisebuchung. Das initiale Ereignis ist die Auslösung eines Reisebuchungsauftrags durch *Kunde*. Dieses Ereignis führt zu einem gekoppelten, synchronen Zustandsübergang der Objekte

*Kunde* und *Agentur* im Rahmen der Durchführung der Transaktion *Reisebuchungsauftrag*. Die diese Transaktion ausführenden Aufgaben *RA>* und *>RA* hinterlassen gleichnamige Zwischenzustände der Objekte *Kunde* und *Agentur*. Die Reisebuchung ist vollständig durchgeführt, wenn die Objekte *Kunde*, *Agentur*, *Fluglinie*, *Autovermietung* und *Hotel* in ihren Endzuständen angekommen sind.

Wird dieser globale Zustand nicht erreicht oder soll die vollständig abgeschlossene Reisebuchung storniert werden (etwa innerhalb vorgesehener Rücktrittsfristen), so müssen die Zustände der an der Durchführung der Geschäftsprozessinstanz beteiligten betrieblichen Objekte in ihre jeweiligen Startzustände zurückgeführt werden. Dies erfolgt durch die Auslösung und Durchführung kompensierender Zustandsübergänge. Jeder dieser kompensierenden Zustandsübergänge wirkt wie eine inverse Transaktion und besteht in der Rückabwicklung eines gekoppelten Zustandsübergangs zweier betrieblicher Objekte. Es ist ausreichend, wenn eines der beiden betrieblichen Objekte den kompensierenden Zustandsübergang auslöst, weil es das andere Objekt aufgrund der synchronen Kopplung „mitnimmt". Wichtig ist, dass das Zustandsschema für jeden gekoppelten Zustandsübergang einen kompensierenden Zustandsübergang bei mindestens einem der beiden Objekte vorsieht. Diese Spezifikation der kompensierenden Zustandsübergänge in geeigneter Form und mit semantischem Bezug zum jeweiligen Geschäftsprozess ist der wesentliche Mehrwert, der durch das Zustandsschema erreicht wird.

| | | |
|---|---|---|
| RB> | >HB | Kunde storniert abgeschlossene Reisebuchung. Innerhalb der Rücktrittsfrist mit der Rückabwicklung beginnen, sonst keine Aktion. |
| >HB | HA> | Hotelbuchung gegenüber dem Hotelier stornieren, ggf. alternativen Hotelier suchen und Prozess fortsetzen. |
| HA> | >MB | Mit der Mietwagenbuchung fortsetzen. |
| >MB | MA> | Mietwagenbuchung gegenüber dem Mietwagenanbieter stornieren, ggf. Alternative suchen und Prozess fortsetzen. |
| MA> | >FB | Mit der Flugbuchung fortsetzen. |
| >FB | FA> | Flugbuchung gegenüber dem Fluganbieter stornieren, ggf. Alternative suchen und Prozess fortsetzen. |
| FA> | >RA | Mit der Reisebuchung fortsetzen. |
| >RA | leer | Reisebuchungsauftrag gegenüber dem Kunden stornieren. |

**Abbildung 7:** Aktionen des Teilsystems *Agentur* zur Rückabwicklung von Reisebuchungsaufträgen

Zum Beispiel ergeben sich für das betriebliche Objekt *Agentur* die in Abbildung 7 gezeigten Anforderungen. Stornierungskosten und -fristen werden hierbei nicht betrachtet. Diese könnten z. B. die Gestaltung des Geschäftsprozesses beeinflussen, z. B. die Reihenfolge von Flug-, Mietwagen- und Hotelbuchung.

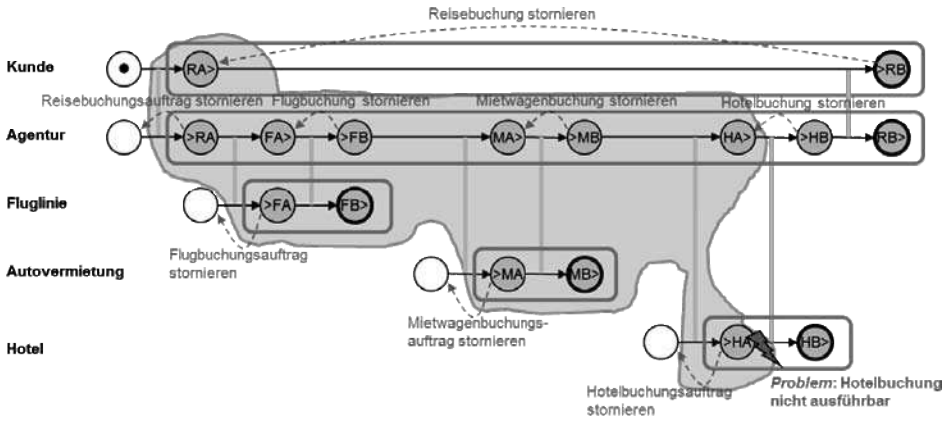


**Abbildung 8:** Geschäftsprozessabbruch im Zustandsschema für das Beispiel Reisebuchung

Angenommen, in einer Instanz des Geschäftsprozesses Reisebuchung ist die Hotelbuchung nicht ausführbar und der Prozess auch nicht mit einer alternativen Hotelbuchung abschließbar, dann ist die Rückabwicklung einzuleiten. Wie Abbildung 8 zeigt, befinden sich zu diesem Zeitpunkt die Objekte *Fluglinie* und *Mietwagen* in ihrem Endzustand, die Objekte *Agentur* und *Hotel* im Zustand *HA>* bzw. *>HA* und das Objekt *Kunde* im Zustand *RA>* nach Erteilung des Reisebuchungsauftrags. Die sich daraus ergebende Kontrollsphäre ist als gefärbte Fläche hinterlegt; sie umfasst die Menge der von der Rückabwicklung betroffenen Zustände.

Dass die Hotelbuchung nicht ausführbar ist, wird vom Objekt *Hotel* erkannt; dieses Objekt löst den kompensierenden Zustandsübergang *Hotelbuchungsauftrag stornieren* aus, wodurch es selbst in den Startzustand, das Objekt *Agentur* in den Zustand *>MB* übergeht. Nun liegt es am Objekt *Agentur*, welches die drei Teilbuchungen koordiniert, den Übergang *Mietwagenbuchung stornieren* auszulösen, wodurch es selbst in den Zustand *MA>*, das Objekt *Autovermietung* in den Zustand *>MA* übergeht. *Autovermietung* löst danach *Mietwagenbuchungsauftrag stornieren* aus, überführt dabei sich selbst in den Startzustand und *Agentur* in den Zustand *>FB*. Es folgt in gleicher Weise die Stornierung der Flugbuchung und schließlich das Stornieren des Reisebuchungsauftrags durch die Agentur gegenüber dem Kunden. Damit sind alle betrieblichen Objekte in Bezug auf die Durchführung der aktuellen Geschäftsprozessinstanz in ihren Startzustand zurückgeführt.

## 5 Diskussion des vorgestellten Ansatzes zur Zustandsmodellierung

Es stellt sich die Frage, welchen Mehrwert die Zustandssicht als „dritte Sicht“ neben der Struktursicht und der Ablaufsicht für die konzeptuelle Modellierung von IS beisteuert. Das Zustandsschema

- visualisiert für die einzelnen betrieblichen Objekte die während der Durchführung eines Geschäftsprozessinstanz auftretenden Zwischen- und Endzustände,
- beschreibt mit semantischem Bezug zum jeweiligen Geschäftsprozessmodell die kompensierenden Zustandsübergänge, die im betrieblichen System notwendig bereitzustellen sind,
- spezifiziert Folgen von kompensierenden Zustandsübergängen, die geeignet sind, um die betrieblichen Objekte aus jedem beliebigen Zustand heraus in einen global konsistenten Zustand rückführen zu können.

Die kompensierenden Zustandsübergänge beschreiben einen wesentlichen Teil der Ausführungssemantik eines Geschäftsprozessmodells und können daher nur mit fachlichem Bezug zu diesem spezifiziert werden. Aus diesem Grund sind die im Zustandsschema zusätzlich bereitgestellten Spezifikationen konzeptueller Natur und fördern die Analyse und Gestaltung der Aufgabenebene eines verteilten IS. Hier ist insbesondere die Verbindung des Konzepts der betrieblichen Transaktion mit dem aus der Datenbanktechnik bekannten ACID-Transaktionskonzept hilfreich.

Darüber hinaus stellt das Zustandsschema Anforderungen an die Gestaltung der Aufgabenträgerebene bereit. Gerade mit Bezug zu serviceorientierten Anwendungssystemen, die ggf. teilweise erst zur Laufzeit aus Diensten konfiguriert werden, kann erwartet werden, dass die damit einhergehende Komplexität ohne konzeptuelle Zustandsmodelle nur schwer beherrschbar sein dürfte. Zum Beispiel ist ein Dienst nur dann in einem verteilten Anwendungssystem einsetzbar, wenn er gleichzeitig die notwendigen Kompensationen bereitstellt. Insofern sollte die konzeptuelle Zustandsmodellierung einen nennenswerten Beitrag zum Entwurf und zur Implementierung zuverlässiger verteilter Anwendungssysteme liefern können.

Zustandsschemata behandeln die Zustandskonsistenz von IS auf der Aufgabenebene. Bezüglich der Gestaltung der Aufgabenträgerebene bestehen erhebliche Freiheitsgrade. Idealerweise sollten die Objekte eines verteilten IS durch transaktionsgeschützte verteilte Anwendungssysteme unterstützt werden. Inwieweit die einzelnen Teilanwendungssysteme unter der Kontrolle eines Transaktionsmanagers ablaufen oder ob Kompensationsfunktionen implementiert werden müssen, inwieweit die Transaktionskonzepte (verteilter) Datenbanksysteme eingesetzt werden usw. sind dabei Fragen der Anwendungssystem-Architektur und der IT-Infrastruktur (z. B. transaktionsorientierte Middleware).

Bezüglich der Spezifikation workflow-basierter betrieblicher Anwendungssysteme ist insbesondere zu untersuchen, welchen Nutzen die Zustandsschemata für die (modellgetriebene) Spezifikation von Workflows bieten. Workflow-Sprachen scheinen hierfür günstige Voraussetzungen zu bieten, da sie eine Differenzierung von Teilsystemen und eine Beschreibung der Interaktion zwischen Teilsystemen unterstützen.

# Literaturverzeichnis


[BPMN11] Business Process Model and Notation (BPMN), Version 2.0, 2011-01-03. http://www.omg.org/spec/BPMN/2.0/PDF/ (Abruf am 2013-10-09)

[Chen76] Chen, P.P.-S.: The Entity-Relationship Model – Toward a Unified View of Data. In: ACM Transactions on Database Systems, Vol. 1, No. 1 (1976), pp. 9-36

[Ens78] Enslow P.H.: What is a ‚Distributed' Data Processing System? In: IEEE Computer, Vol. 11, No. 1, January 1978, pp. 13-21

[FeSi90] Ferstl, O.K.; Sinz E.J.: Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: WIRTSCHAFTSINFORMATIK 32 (1990) 6, S. 566-581

[FeSi91] Ferstl, O.K.; Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In WIRTSCHAFTSINFORMATIK 33 (1991) 6, S. 477-491

[FeSi95] Ferstl, O.K.; Sinz E.J.: Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. In: WIRTSCHAFTSINFORMATIK 37 (1995) 3, S. 209-220

[FeSi13] Ferstl, O.K.; Sinz E.J.: Grundlagen der Wirtschaftsinformatik. 7. Auflage, Oldenbourg, München 2013

[GrRe93] Gray, J.; Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, California 1993

[LoSc87] Lockemann, P.C.; Schmidt J.W.: Datenbank-Handbuch. Springer, Berlin 1987

[Nütt13] Nüttgens, M.: EPK. In: Kurbel, Karl; Becker, Jörg; Gronau, Norbert; Sinz, Elmar; Suhl, Leena (Herausgeber): Enzyklopädie der Wirtschaftsinformatik – Online-Lexikon. Siebte Auflage. München : Oldenbourg, 13.9.2013. http://www.enzyklopaedie-der-wirtschaftsinformatik.de (Abruf: 2013-10-08).

[OAS07] OASIS Web Services Business Process Execution Language Version 2.0, OASIS Standard, 11 April 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf (Abruf am 2013-10-09)

[Reut87] Reuter, A.: Maßnahmen zur Wahrung von Sicherheits- und Integritätsbedingungen. In [LoSc87], S. 390-479

[Reis10] Reisig, W.: Petri-Netze. Modellierungstechnik, Analysemethoden, Fallstudien. Vieweg und Teubner, Wiesbaden 2010

[Sinz88] Sinz E.J.: Das Strukturierte Entity-Relationship-Modell (SER-Modell). In: Angewandte Informatik, Band 30, Heft 5 (1988), S. 191-202

[Tiwa11] Tiwari S.: Professional NoSQL. Wiley, Indianapolis 2011

[UML11] OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1, 2011-08-06. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/ (Abruf am 2013-10-09)