Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 2481

Realisierung von verteilten Anwendungsszenarien anhand einer Service-orientierten Architektur

Patrick M. Schneider

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr.-Ing. habil. B. Mitschang
Betreuer:	Dr.-Ing. Dipl.-Inform. T. Specht Dipl.-Inf. C. Dorda
begonnen am:	1. April 2006
beendet am:	2. Oktober 2006
CR-Klassifikation:	C.0.0, C.1.3, C.2.4, D.2.11, H.4.1

Inhaltsverzeichnis

Abkürzungsverzeichnis III

Abbildungsverzeichnis IV

1 Einleitung 1

2 Problemstellung 3
2.1 Randbedingungen . . . 4
2.1.1 Historische Entwicklung der Softwarearchitekturen . . . 4
2.1.2 Allgemeine Randbedingungen bei der Softwareintegration . . . 6
2.1.3 E-Government spezifische Randbedingungen . . . 7
2.2 Zielsetzung der Arbeit . . . 10

3 Stand der Technik 11
3.1 Geschäftsprozessmodellierung . . . 11
3.1.1 ARIS Methodik . . . 13
3.1.2 Integration Engineering Methodik . . . 15
3.1.3 Petrinetze . . . 17
3.1.4 Aktivitätsdiagramme . . . 18
3.2 Service-orientierte Architektur . . . 19
3.2.1 Web Services . . . 22
3.2.2 WSDL 1.1 . . . 23
3.2.3 SOAP . . . 25
3.2.4 Universal Data Description and Discovery (UDDI) . . . 25
3.2.5 WS-Security . . . 26
3.2.6 WS-Coordination . . . 27
3.3 Workflowmanagement Systeme (WfMS) . . . 28
3.3.1 BPEL . . . 29
3.4 Vorgehensmodelle für Softwareentwicklung . . . 39
3.4.1 Das Spiralmodell . . . 40
3.4.2 Der Rational Unified Process . . . 40
3.4.3 Das Phasenmodell nach Molters . . . 41

4 Vorgehensweise 42
4.1 Anforderungen an das Vorgehensmodell . . . 43
4.2 Das Phasenmodell der Prozessintegration . . . 45
4.2.1 Geschäftsprozess-analyse / definition . . . 46
4.2.2 IT-Modellierung . . . 47
4.2.3 Komponenten-erstellung/erweiterung . . . 48
4.2.4 Komponententest . . . 49
4.2.5 Komponentenintegration . . . 49
4.2.6 Integrationstest . . . 50
4.2.7 Betrieb . . . 51
4.2.8 Monitoring/Benchmarking . . . 51

4.2.9 Optimierung . . . 52
4.3 Modelltransformation . . . 53
4.3.1 Anforderungen an den Kontrollfluss . . . 54
4.3.2 1. Schritt: Bereinigung des Ausgangsmodells . . . 57
4.3.3 2. Schritt: Kontrollflussanalyse . . . 61
4.3.4 3. Schritt: Identifikation von relevanten Interaktionsschemata . . . 70
4.3.5 Namensgebung . . . 73
4.3.6 Änderungen an der „Integration Engineering“ Methodik . . . 75
4.4 Anforderungen an die Geschäftsprozessmodellierung . . . 76

5 Umsetzung **79**
5.1 Ausgangslage . . . 79
5.2 Geschäftsprozessmodellierung . . . 80
5.3 IT-Modellierung . . . 84
5.4 Komponentenentwicklung . . . 85
5.5 Komponententest . . . 87
5.6 Komponentenintegration . . . 88
5.6.1 Anforderungen an den Kontrollfluss . . . 88
5.6.2 1. Schritt: Bereinigung des Ausgangsmodells . . . 88
5.6.3 2. Schritt: Kontrollflussanalyse . . . 89
5.6.4 3. Schritt: Identifikation der relevanten Interaktionsschemata . . . 90
5.7 Weitere Phasen . . . 92

6 Zusammenfassung - Fazit **93**

7 Literaturverzeichnis **95**

Abkürzungsverzeichnis

AS	**A**pplication **S**erver
BPEL	**B**usiness **P**rocess **E**xecution **L**anguage
BPO	**B**usiness **P**rocess **O**ptimization
BPR	**B**usiness **P**rocess **R**eengineering
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
DBMS	**D**aten**b**ank **M**anagement **S**ystemen
DMS	**D**ocument **M**anagement **S**ystem
DVDV	**D**eutsches **V**erwaltungs und **D**ienste**V**erzeichnis
EAI	**E**nterprise **A**pplication **I**ntegration
ECA	**E**vent **C**ondition **A**ction
eFZD	**e**rweiterte **F**unktions**Z**uordnungs**D**iagramme
EPK	**E**reignisgesteuerte **P**rozess **K**etten
IDL	**I**nterface **D**efinition **L**anguage
OSCI	**O**nline **S**ervices **C**omputer **I**nterface
RUP	**R**ational **U**nified **P**rocess
SME	**S**mall and **M**ediumsized **E**nterprizes
SOA	**S**ervice **O**riented **A**rchitecture
SOAP	ehemals **S**imple **O**bject **A**ccess **P**rotocol heute nur noch Name
UML	**U**nified **M**odelling **L**anguage
WfMS	**W**ork**f**low**m**anagement **S**ystem
WS	**W**eb **S**ervice
WSDL	**W**eb **S**ervice **D**escription **L**anguage
XML	e**X**tensible **M**arkup **L**anguage
XSD	**X**ML **S**chema **D**efinition

Abbildungsverzeichnis

1 Das magische Dreieck: Zeit-Kosten-Qualität . . . 3
2 Typische Anwendungsarchitekturen . . . 5
3 Klassifikation der verschiedenen Partnerschaften nach [Bun05c] . . . 7
4 Das OSCI Transportverfahren [Din05] . . . 9
5 Das ARIS house of business engineering [SNZ98] . . . 13
6 Beispiel einer EPK . . . 14
7 Beispiel eines eFZD . . . 17
8 Petrinetz Beispiel . . . 18
9 Beispiel Lagerauffüllen (UML 2.0) . . . 19
10 Das SOA Dreieck nach [WCL+05] . . . 21
11 a) Der SOA Stack [WCL+05] b) spezialisiert auf WS Standards . . . 21
12 Aufbau der WSDL 1.1 . . . 24
13 SOAP Nachrichtenstruktur . . . 25
14 Das UDDI Datenmodell nach [OAS05] . . . 26
15 Übersicht evaluierter WfMS . . . 29
16 Veranschaulichung des flow Konstruktes . . . 38
17 Der Rational Unified Process [KK03] . . . 40
18 Lifecycle Modell [SJW05] . . . 41
19 Abgewandeltes Phasenmodell Prozessorientierter Integration nach [SJW05] 45
20 Beispiel einer EPK . . . 47
21 Die aus diesem Prozessschritt entstanden Diagrammtypen . . . 48
22 Schemenhafte Darstellung des Transformationsalgorithmus . . . 50
23 Der Transformationsalgorithmus grafisch dargestellt . . . 53
24 Beispielhafte Darstellung relevanter Modellbausteine . . . 54
25 Beispiel für die aufgeführten Definitionen . . . 55
26 Beispielgraphen . . . 57
27 Beispiel für Bedingungen in Ereignissen . . . 58
28 Linearer Kontrollfluss in einer EPK . . . 61

29 Beispiel für eine Verzweigung in einer EPK 63
30 Mehrere mögliche Initialereignisse 65
31 Nicht abbildbares Konstrukt [Kop05] 67
32 Eine EPK, die eine Schleife mit zwei Ausgängen darstellt 68
33 Klassische while Schleife 69
34 Do-while Schleife 69
35 Repeat-until Schleife 69
36 Receive Aktivität in einem eFZD 71
37 Reply in einem eFZD modelliert 72
38 Invoke in einem eFZD modelliert 72
39 Synchrones invoke als eFZD modelliert 73
40 Die neu eingeführten Symboltypen 75
41 Beispielhafte UML Sequenzdiagramme 76
42 Mehrere Eingänge einer Schleife 77
43 Die einfache Melderegisterauskunft als EPK modelliert 81
44 „Anfrage stellen“ eFZD * 82
45 „Zentrale Datenbank abfragen“ eFZD * 82
46 „Plausibilität prüfen“ eFZD * 83
47 „Melderegister durchsuchen“ eFZD * 83
48 „Intermediär Antwort versenden“ eFZD * 84
49 „Antwort übermitteln“ eFZD * 84
50 „Ergebnis anschauen“ eFZD * 84
51 Das resultierende Anwendunssystem Diagramm 85
52 Die Weboberfläche des WebServices 86
53 Bildschirmabzug „Beantragung einer eMRA“ 86
54 Bildschirmabzug „Antwort auf eMRA“ 87
55 Das bereinigte fachliche Modell 88

1 Einleitung

Die Wertschöpfungsketten moderner Unternehmen sind heutzutage stetigen Veränderungen ausgesetzt. Mit ihnen verändern sich zwangsweise auch die Geschäftsprozesse, die diese abbilden sollen. Durch den Druck des Marktes und der Globalisierung entstehen für Unternehmen neue Anforderungen an ihre Organisationsstrukturen und Abläufe. Eine sehr wichtige solche Anforderung ist, auf Veränderungen des Marktes möglichst rasch zu reagieren, um die Vorteile, die sich durch einen schnellen Markteintritt ergeben, nutzen zu können. In diesem Umfeld müssen sich natürlich auch die IT Infrastrukturen den neuen Gegebenheiten anpassen. Ein Weg, auf die sich ständig verändernden Anforderungen an die IT-Systeme zu reagieren, sind so genannte Service-orientierte Architekturen. Das Thema Service-orientierte Architektur hat in den letzten 5 Jahren immer mehr an Bedeutung gewonnen. Es ist geradezu zu einem Trend in der IT und Geschäftswelt geworden. Ein Grund für den Siegeszug von Service-orientierten Architekturen ist die Flexibilität, die sie für die Anwendungserstellung bieten. Dabei sind die Konzepte, die mit einer SOA in Verbindung gebracht werden, genauso weit verteilt, wie die Anwendergruppen, die diese Termini gebrauchen. Um etwaige Missverständnisse von vornherein auszuräumen, werden in Kapitel 3 die verwendeten Techniken näher beschrieben.

Innerhalb der letzten 20 Jahre „wuchs“ eine unglaubliche Vielfalt an heterogenen Systemen in größeren Unternehmen, die eine sinnvolle und einfache Zusammenarbeit der Anwendungssysteme fast unmöglich machten. Selbst wenn innerhalb eines Unternehmens die Anwendungslandschaft standardisiert, und eine Integration der einzelnen Softwarekomponenten ohne zu große Schwierigkeiten möglich war, stellte die unternehmensstrategische Entscheidung, einen Mitbewerber bzw. ein Konkurenzunternehmen aufzukaufen, und die damit verbundene Anwendungslandschaft in das Unternehmen einzugliedern, die IT-Abteilungen regelmäßig vor große Probleme.

In vielen Führungsetagen werden fachliche Geschäftsprozessmodelle zur Dokumentation der leistungserstellenden / wertschöpfenden Prozesse angefertigt. Um verschiedene Anwendungssysteme zu einem dem Prozess optimal angepassten Anwendungssystem zu integrieren, ist das Wissen aus diesem fachlichen Modell unabdingbar. Jedoch besteht zwischen dem fachlichen und dem technischen Modell eine Lücke. Dies ist durch die verschiedenen Abstraktionsebenen bei der Sichtweise auf den Prozess begründet. Würde man einem Entscheider eines Unternehmens einen technischen Prozess vorlegen, der auf dessen fachlichen Modell basiert, wäre er ob der vielen, für ihn unwichtigen Details verunsichert. Diese Lücke zwischen fachlichem und technischem Modell soll im Rahmen dieser Arbeit verringert werden. Dies soll auf Basis der „Integration Engineering“ Methodik geschehen, die analysiert, evaluiert und wenn nötig erweitert werden soll. Dabei ist eines der Schwerpunkte die Transformation von einem fachlichen Modell, das anhand der „IE“ Methodik erstellt wurde, zu einem technischen Modell mit Hilfe von BPEL. Um dies in der Praxis zu erkunden, wird sie anhand eines Szenarios aus dem E-Government Umfeld analysiert und evaluiert.

Aufbau der Arbeit

Kapitel 2 verdeutlicht die Problemstellungen bei der Integration von verteilten Anwendungssystemen. In diesem Kapitel werden auch die Besonderheiten des betrachteten E-Government Szenarios dargestellt. Daraus wird dann die Zielstellung dieser Arbeit abgeleitet und vorgestellt.

Kapitel 3 beschreibt die eingesetzten Technologien und Standards ausführlich.

Im Kapitel 4 wird die Methodik vorgestellt, die für die Umsetzung zu einem technischen Geschäftsprozessmodell angewandt wurde.

In Kapitel 5 wird dann anhand der konkreten Szenarien die Umsetzung der in Kapitel 4 vorgestellten Methodik beschrieben.

Zum Schluss werden in Kapitel 6 die Schwierigkeiten und Probleme beschrieben, die bei einer Integration auftreten, sowie ein Ausblick über weiteren Forschungsbedarf gegeben.

2 Problemstellung

Die Integration komplexer heterogener Anwendungssysteme zu neueren, den Geschäftsprozessen besser angepassten Computersystemen hat in den letzten 5 Jahren große Fortschritte gemacht. Aus der Not heraus geboren, die heterogene Anwendungslandschaft, die sich über die letzten 40 Jahre Computerunterstützung in großen Firmen und Behörden aufgebaut hat, effektiver miteinander zu nutzen, hat sich das sogenannte Workflowmanagement in der Zwischenzeit auch für die Führungsetagen zu einem Werkzeug entwickelt, das einen tieferen Einblick in die bestehenden Geschäftsprozesse liefert. Jedoch muss man in diesem Zusammenhang erwähnen, dass es eine signifikante Lücke der Modelle zwischen der technischen Sicht auf IT-gestützte Geschäftsprozesse und der fachlichen Sicht auf die Geschäftsprozesse aus betriebswirtschaftlicher Sicht gibt[FTH05]. Demnach haben sich in der Betriebswirtschaft einige Methoden zur Geschäftsprozessmodellierung entwickelt deren Fokus nicht auf der technischen Sicht der Prozesse liegt. Somit existieren heute zwei verschiedene Sprachtypen nebeneinander. Einerseits Prozessbeschreibungssprachen wie beispielsweise ereignisgesteuerte Prozessketten, deren Fokus auf dem Ablauf und der Dokumentation betrieblicher Prozesse liegt. Andererseits Prozessbeschreibungssprachen wie die Sprache BPEL oder XLANG, deren Fokus wiederum eher auf der technischen Seite der Geschäftsprozesse liegt. Um sinnvolle, „lebendige“ Modelle für alle Ebenen des Geschäftslebens zu bekommen, müssen beide Sorten von Sprachen erweitert werden. Oder Transformationsengines bereitgestellt werden, die die Lücke zwischen beiden Modellen schließen und eine Benutzung der Modelle in allen Ebenen, von der technischen bis zur betriebswirtschaftlichen, ermöglichen.

Unternehmen sehen sich heutzutage vielen Herausforderungen ausgesetzt, denen sie, um am Markt erfolgreich bestehen zu können, möglichst flexibel entgegentreten müssen. Durch den Konkurrenzdruck, der durch die Globalisierung auf die Unternehmen entsteht, wird die Zeit zu einem immer kritischeren Faktor. Da Europa nicht mit den Löhnen der Schwellenländer konkurrieren kann, ist ein weiterer kritischer Faktor die Qualität, die ein erstelltes Produkt oder eine Dienstleistung hat. In diesem Zuge wird versucht, die Barrieren des so genannten „magischen Dreiecks“, das den Tradeoff zwischen Kosten, Qualität und Zeit darstellt (s. **Abbildung 1**), zu durchbrechen.

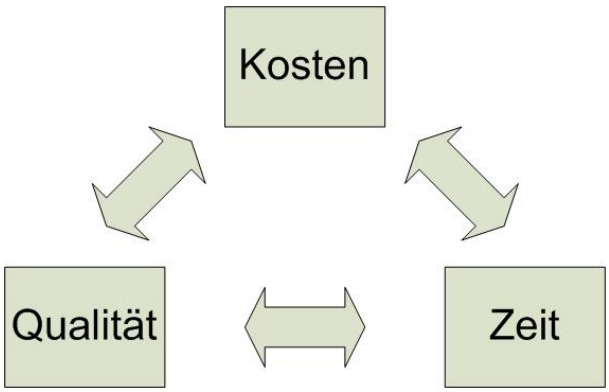


Abbildung 1: Das magische Dreieck: Zeit-Kosten-Qualität

Eine Möglichkeit an die Grenzen des „magischen Dreiecks“ zu gelangen, ist die kontinuierliche Verbesserung der wertschöpfenden Prozesse. Dabei wird durch einen standardisiert ablaufenden Prozess die Qualität gewahrt, durch optimal zusammenspielende Teilaktivitäten die nötige Zeit minimiert und dadurch wiederum die Kosten gesenkt. Daher wird im betriebswirtschaftlichen Umfeld in den letzten Jahren in zunehmendem Maße eine Abkehr von der rein aufbauorientierten Organisation hin zur ablauforientierten beziehungsweise prozessgetriebenen Organisation vollzogen.

2.1 Randbedingungen

Im Folgenden Kapitel werden die Randbedingungen bei einer Anwendungsintegration geklärt. Die Randbedingungen, die bei der Integration von Anwendungssystemen zum tragen kommen, sind zu einem großen Teil auf die geschichtliche Entwicklung von Softwarearchitekturen zurückzuführen.

2.1.1 Historische Entwicklung der Softwarearchitekturen

Die Entwicklung der Informations- und Kommunikationstechnologie in den letzten 50 Jahren war atemberaubend. Ein Indiz dafür ist beispielsweise „Moore's Law“. Der Intel Mitbegründer Gordon E. Moore formulierte 1965 in einem Artikel in der Fachzeitschrift „Electronics“ diese sehr bekannte These. Die Kernaussage dieser These ist, dass sich die Anzahl der Transistoren in einer integrierten Schaltung jedes Jahr verdoppelt. Diese empirische Beobachtung hat bis heute in etwas abgschwächter Form Bestand.

Ebenso wie sich in diesem Zeitraum die Hardware entwickelt hat, entwickelte sich analog auch die Software, mit allen Konsequenzen, die sich daraus ergeben.

So wurde die Anwendungslandschaft in einem größeren Unternehmen bis in die 80er Jahre meist durch sogenannte Mainframes geprägt. Diese Mainframes waren zentrale Recheneinheiten, auf denen die komplette Software eines Unternehmens lief. Wenn neue Funktionalität gewünscht wurde, oder Funktionen von Programmen zu komplexeren Funktionen aggregiert werden sollten, erleichterte dieser Fakt die Arbeit eines Systemintegrators erheblich, da er bei einer Anwendungsintegration immer dieselben Gegebenheiten vor fand. So waren das Betriebssystem und die Hardware für alle auf dem Mainframe laufenden Programme dieselben. Meist wurden sogar die vorhandenen proprietären Softwarelösungen mit ein und derselben Programmiersprache erstellt.

Mit dem Siegeszug der Personal Computer (PC's) Anfang der 90er Jahre geriet diese Art der Monolithischen Systemarchitektur ins Wanken. Es erschienen immer mehr individuelle Softwarelösungen. In dieser Zeit entstand eine Vielzahl von Anwendungen, die den einzelnen Mitarbeiter in ihrer täglichen Arbeit unterstützen sollten. So wurden die Entscheidungsträger eines Unternehmens durch Anwendungssysteme unterstützt, die ihnen Zugang zu wichtigen Unternehmensinformationen gaben. Dazu zählen beispielsweise Rechnungswesen-, Controlling oder Management-Informationssysteme. Um die wertschöpfenden Kernprozesse zu unterstützen, wurden Anwendungssysteme wie

Produktionsplanungs-, Einkaufs- und Steuerungs-Systeme implementiert oder von verschiedenen, spezialisierten Herstellern eingekauft. Jedoch nicht nur die Anzahl der verschiedenen Anwendungssysteme stieg, sondern ebenso die Zahl der eingesetzten Betriebssysteme nahm kontinuierlich zu. So werden heute auf Servern ein und desselben Unternehmens Betriebssysteme wie Microsoft Windows, Solaris/Unix von Sun, OS/2 von IBM und Linux verteilt betrieben. Auf diesen Servern laufen zentrale Datenbanken, die die gemeinsame Datengrundlage für verteilt laufende Anwendungen darstellen.

Um sinnvoll Anwendungssysteme integrieren zu können, ist eine detaillierte Analyse der verwendeten Architekturen nötig. Mainframes (vgl. monolithische Systemarchitektur) beherbergten ein komplettes Anwendungssystem. Angefangen von der Benutzeroberfläche, über die für das Programm nötige Logik, sowie die vom Programm genutzte Datenhaltung waren alle „Schichten“ auf ein und demselben Rechner beherbergt. Die Benutzeroberfläche wurde meist auf sogenannten „dumb-Terminals“ ausgegeben.

Im Laufe der Zeit und mit zunehmender Rechenleistung der Einzel-Arbeitsplätze hielt eine andere Art der Systemarchitektur Einzug. Um die Rechenleistung der Arbeitsplatzrechner mehr auszulasten und dabei auch die Server zu entlasten, trennte man bei der so genannten 2-Schichten Architektur die Benutzeroberflächen „Logik“ von der Applikations- und Datenhaltungs-Schicht. Dies war die erste Art der „Client-Server“ Architektur.

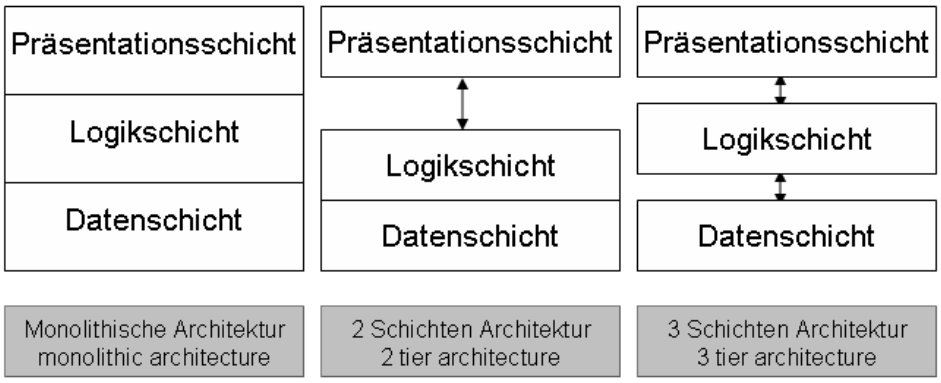


Abbildung 2: Typische Anwendungsarchitekturen

Da die Rechenleistung der Arbeitsplatzrechener immer weiter stieg und der Gedanke der Wiederverwendung von Systemteilen eine immer größere Rolle spielte, unterteilte man die Logik und Datenhaltungsschicht weiter in eine separate Datenhaltungs- und eine Logikschicht. Die Datenhaltungsschicht wird meist von Datenbank Management Systemen (DBMS) übernommen. Die Logikschicht wird meist von sogenannten Applikation Servern (AS) beherbergt . Auf ihnen konnten Business Objekte angelegt werden, die es erlaubten, logisch zusammenhängende Geschäftsfunktionen, unabhängig von ihrer Präsentationsschicht und der dazugehörigen Datenhaltungsschicht, zu implementieren. Dies war ein sehr wichtiger Schritt in Richtung Wiederverwendung. Diese Art der Architektur wird

Drei Schichten Architektur genannt. Um nun verschiedene Applikationsserver zu integrieren und daraus komplexere Geschäftsfunktionen zu erstellen, bedarf es einer n-Schichten Architektur mit zusätzlichen Integrationsschichten. Wie in **Abbildung 2** dargestellt, wandelten sich die verwendeten Architekturen also von streng „monolithischen“ (vgl. Mainframes) über eine 2-Schicht Architektur, bis hin zur heute weitverbreiteten 3-Schicht Architektur.

2.1.2 Allgemeine Randbedingungen bei der Softwareintegration

Die meisten Anwendungslandschaften, die größere Unternehmen und Behörden heutzutage unterhalten, stellen ein Problem für die Integration der beteiligten Systeme dar. Die verwendeten Technologien sind historisch gewachsen und unterscheiden sich in ihren Funktionsweisen und in ihren Schnittstellen doch meist sehr deutlich. Dies ist hauptsächlich historisch begründet. Daher wurde in Abschnitt 2.1.1 ein kleiner Einblick in die historische Entwicklung von Anwendungssystemen und Architekturen gegeben.

Die Anforderungen, die an die Integration von verteilten Anwendungssystemen gestellt werden, unterscheiden sich meist kaum von den Anforderungen an die schon realisierten Teilsysteme. Wichtige Randbedingungen für eine Integrationslösung sind beispielsweise:

- Kosten
- Datensicherheit
- Ausfallsicherheit
- Skalierbarkeit
- Benutzbarkeit

Die integrierten Teilsysteme erfüllen die gestellten Anforderungen meist, und sollten sie dieses nicht tun, muss auch von einer Integration abgesehen werden. In diesem Fall sollte die Teilkomponente, die den Anforderungen nicht mehr entspricht, ersetzt werden. Bei der Integration von verteilten Anwendungssystemen gibt es verschiedene Ebenen, auf denen integriert werden kann. Wie in Abschnitt 2.1.1 beschrieben, bestehen Architekturen aus verschiedenen Schichten. Grundsätzlich ist es möglich, auf jeder dieser Schichten eine Integration durchzuführen.

Im folgenden Fall wird davon ausgegangen, das genau drei Schichten existieren. Eine Präsentationsschicht, eine Logikschicht und eine Datenschicht. Anhand von Adaptoren, die pro Schicht in 2 Kategorien unterteilt werden, können Statusänderungen der zu integrierenden Applikation erfasst werden. Unterschieden wird hierbei zwischen In- beziehungsweise Out-Adaptoren. Out-Adaptoren leiten eine Statusveränderung der Anwendung an einen „Conroller“ weiter, um darauf reagieren zu können. In-Adaptoren werden dazu verwendet, aufgrund externer Ereignisse eine Applikation zu einer Satusveränderung zu bewegen.

2.1.3 E-Government spezifische Randbedingungen

Da in dieser Arbeit anhand eines Praxisbeispiels aus dem E-Government-Umfeld eine Methodik zur Integration von verteilten Anwendungssystemen analysiert und angepasst werden soll, werden in den folgenden Abschnitten einige Randbedingungen geschildert, die bei einer Integration im betriebswirtschaftlichen Umfeld nicht gegeben sind.

2.1.3.1 Definition E-Government Die Verwendung des Begriffs E-Government ist oft nicht eindeutig. Oft wird nicht klar was darunter zu verstehen ist und welche Chancen sich daraus ergeben. Oftmals wird es nur als weiteres Schlagwort des Computerzeitalters genannt. Eine andere Sichtweise sieht darin den nächsten logischen Schritt der Verwaltungsinformatik oder die elektronische Ausprägung der bisherigen Bemühungen zur Verwaltungsreform in den Behörden. Folgt man der Bundesinitiative BundOnline 2005 [Bun05b], umfasst Electronic Government alle Prozesse der Entscheidungsfindung und Leistungserstellung in Politik, Staat und Verwaltung, soweit diese unter weitestgehender Nutzung von Informations- und Kommunikationstechnologien stattfinden. Diese bieten dabei sehr vielfältige Nutzungsmöglichkeiten. Sie fangen an bei der Verwaltungsmodernisierung durch elektronische Vorgangsbearbeitung, reichen über die Bereitstellung von Verwaltungsinformationen auf Behörden-Portalen im Internet bis hin zu den komplexen Transaktionen und interaktiven elektronischen Bürgerdiensten im Netz. Ziel des E-Government ist es, externen Verwaltungskunden, gleich ob Bürgerinnen und Bürgern, Wirtschaftsunternehmen oder anderen Verwaltungen, sämtliche Behördendienstleistungen elektronisch zugänglich zu machen. E-Government steht also für den öffentlichen Sektor in der sich entwickelnden Informationsgesellschaft.

Wie in der „E-Business Welt“ wird E-Government in verschiedene Bereiche unterteilt. Diese Klassifikation wird in **Abbildung 3** dargestellt. Anwendungen, die darauf ausgerichtet sind, dem Bürger einen Dienst zur Verfügung zu stellen, werden G2C (Government to Citizen) genannt. Anwendungen, die Funktionalität hauptsächlich für Unternehmen bereit stellen, werden als G2B (Government to Business) bezeichnet. Als letztes bleiben noch die Anwendungsfeleder, die die Interaktion von Behörden untereinander realisieren. Diese werden unter G2G (Government to Government) subsumiert.

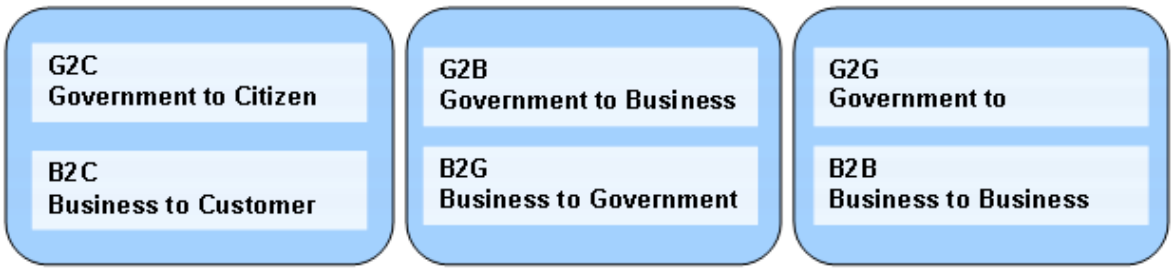


Abbildung 3: Klassifikation der verschiedenen Partnerschaften nach [Bun05c]

2.1.3.2 Gesetzliche Grundlagen Im Rahmen des E-Government stellen die gesetzlichen Grundlagen eine Art Grundanforderungen an die realisierten Anwendungssysteme. Die in diesem Szenario relevanten Gesetze sind folgende:

- Bundesdatenschutzgesetz (BDSG)[Bun03]
- Melderechtsrahmengesetz (MRRG) [Bun05a]
- Landesmeldegesetz des Landes Mecklenburg-Vorpommern [Lan07]
- Signaturgesetz (SigG) [Bun01]

Die Grundlage für gesetzeskonforme Interaktion mit Behörden über das Internet sind durch das Signaturgesetzt geregelt. Darin wird beispielsweise festgelegt welche Voraussetzungen für eine rechtsgültige „Unterschrift“ im Internet gelten müssen. In dieser Arbeit kam hauptsächlich das Landesmeldegesetz des Landes Mecklenburg-Vorpommern zum Einsatz, das für die Modellierung der einfachen Melderegisterauskunft die Anforderungen stellt. Basis für das Landesmeldegesetz des Landes Mecklenburg-Vorpommern ist das Melderechtsrahmengesetz.

2.1.3.3 OSCI Transport Bei der Kommunikation sensibler Daten zwischen ein oder mehreren Partnern ist eines der wichtigsten Themen die Sicherheit und Vertraulichkeit der Daten. Es gibt 4 Punkte, die in diesem Zusammenhang von der OSCI Leitstelle Bremen [OSCa] erarbeitet wurden und mit den dazugehörigen Sicherheitsrisiken unten aufgeführt werden.

- **Vertraulichkeit** - *Preisgabe von vertraulichen Informationen an Unbefugte*
- **Integrität** - *Verfälschung von Informationen durch unbefugte Dritte*
- **Authentizität** - *Vorspiegelung von Identitäten durch Dritte*
- **Nachweisbarkeit** - *Abstreiten von erfolgten Kommunikationsbeziehungen*

Da im Jahre 2001, nach Ansicht des mit der Standardisierung beauftragten Arbeitskreises, eine ausreichende Unterstützung der bis dato am Markt erhältlichen Lösungen und Standards diese Kriterien nicht ausreichend gewährleisteten, wurde ein eigener Standard erarbeitet. Kernpunkte des Verfahrens sind gängige Verfahren aus der IT Sicherheitstechnik. So wird den oben genannten Aspekten innerhalb des OSCI Transport Verfahrens durch folgende Verfahren entgegnet.

- **Vertraulichkeit** - Asymmetrische Verschlüsselungsmethoden (RSA/DSA)
- **Integrität** - Elekronische Signaturen (sichere Hashfunktionen + Kryptierung)
- **Authentizität** - Qualifizierte/Fortgeschrittene Signaturen (Personen/Programme)

- **Nachweisbarkeit** - Revisionssicheres Protokoll Qualifizierte Zeitstempel

OSCI Transport basiert als Nachrichtenprotokoll auf dem aus dem Web Service Umfeld bekannten und in Kapitel 3.2.3 beschriebenen SOAP Protokoll. Für mehr technische Details sowie eine Spezifikation wird an dieser Stelle auf [OSCb] verwiesen. Ähnlich wie beim WS-Security Standard(Kapitel 3.2.5), definiert OSCI Transport verschiedene Header Felder in denen die verwendeten Verschlüsselungsverfahren sowie Zertifikate abgelegt sind.

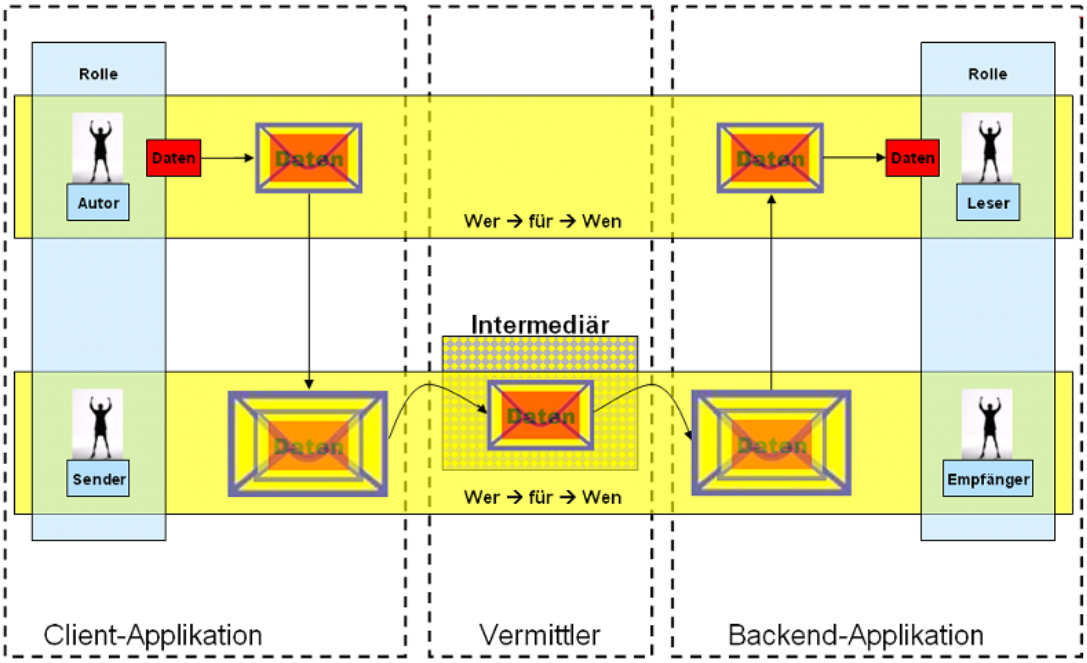


Abbildung 4: Das OSCI Transportverfahren [Din05]

Wie in **Abbildung 4** zu sehen ist, stellt der so genannte Intermediär den Kernpunkt des OSCI Transport Protokolles dar. Um eine sinvolle Verarbeitung zu garantieren, wird eine doppelte Verschlüselung angewandt. Sie unterscheidet zwischen den Transportdaten (wie beispielsweise Adresse des Empfängers etc.) und den Nutzdaten, die nur für den Empfänger zu lesen sein sollen. Der Standardablauf einer Kommunikation von Autor A an einen Empfänger B funktioniert folgendermaßen: Zuerst erstellt der Autor A die zu sendenden Daten. Danach verschlüsselt er seine Nutzdaten mit seinem Privat Key (PK) und dem Public Key des Empfängers B. Als dritter Schritt werden die transportrelevanten Daten für die zwischengelagerten Intermediäre verschlüsselt. Abschließend wird das doppelt verschlüsselte Nachrichtenpaket an den Intermediär abgschickt. Das Auffinden und Abrufen der relevanten Schlüssel und Zertifikate wird anhand eines mit einer UDDI (vgl. Kapitel 3.2.4) vergleichbaren, so genannten DVDV erreicht (**D**eutsches **V**erwaltungs und **D**ienste**V**erzeichnis). Sinn und Zweck des Intermediärs ist die Prüfung der Zertifikate und die Bereitstellung von sogenannten Postfächern, in denen die Nachrichten für die Empfänger sicher zwischengelagert werden. Aus diesem Grund werden Intermediäre auch als Virtuelle Poststellen (VPS) bezeichnet.

2.1.3.4 OSCI XMeld OSCI XMeld ist ein Standard, der im Zuge der Initiative „Bund Online 2005“ [Bun05b] entstanden ist. Die Möglichkeiten, die moderne Informations und Kommunikationsmittel bereitstellen, bieten auch dem Staat Möglichkeiten, Leistungen, die er für die Bürger erbringt, über andere, neue Kanäle den Nachfragern zur Verfügung zu stellen. So nutzen heute schon über 60 % der Bundesbürger über 18 Jahren das Internet. Ende 2003 wurden daher durch den Bund und die Länder einige Dienstleistungen des Staates festgelegt, welche als erstes dem Bürger über die neuen Medien zur Verfügung gestellt werden sollten.

Das Meldewesen ist eines der zentralen Verwaltungsaufgaben, die der Staat übernimmt. Daher wurde diese Domäne zusammen mit der Justiz als erstes dazu auserkoren, eine Vorreiterrolle des E-Gvernments zu spielen. Inhaltlich regelt der XMeld Standard die Datenformate, die in der Domäne des Meldewesens benutzt werden. Er definiert alle für das Betreiben von Melderegistern nötigen Datentypen. Die Beschreibungen der Datenformate werden als XSD Dateien vorgehalten und können über die von der OSCI Leitstelle in Bremen betriebene Homepage [OSCa] heruntergeladen werden. Die neueste Version dieses Standards ist OSCI XMeld 1.3.

2.2 Zielsetzung der Arbeit

Die Vorgehensweise zur Integration von verteilten Anwendungssystemen stellt den Schwerpunkt der in dieser Arbeit behandelten Thematik dar. Dabei verursacht in diesem Zusammenhang die Heterogenität der an einem Geschäftsprozess beteiligten Anwendungssysteme erhebliche Probleme. Wie in dem vorigen Abschnitt dieses Kapitels beschrieben, fordern Geschäftsprozesse eine hohe Flexibilität der sie unterstützenden Anwendungssysteme, da sie einem konstanten Wandel unterzogen sind. Eine Service Orientierte Architektur bietet eine solche Flexibilität und wird in dieser Arbeit angewendet.

Ziel dieser Arbeit ist es, die im Forschungsprojekt „Integration Engineering“ (IE) [FTH05] entwickelte Methodik zur Integration von verteilten Anwendungssystemen anhand eines Praxisbeispiels aus dem E-Government Umfeld zu analysieren. Dabei sollte so wenig wie möglich Redundanz bei der Modellierung auftreten. Konkret heißt dies, Änderungen, die an einem Geschäftsprozessmodell vollzogen werden, sollten leicht in ein technisches Modell durchpropagiert werden können. Daher soll im Rahmen dieser Arbeit die theoretische Grundlage für eine Transfomation eines durch die IE Methode modellierten Geschäftsprozesses in ein technisches Prozessmodellgerüst, das anhand der „Business Process Execution Language for Web Services“ (BPEL4WS) realisiert wird, erarbeitet werden. Dazu gehören speziell die Vorausetzungen, die gelten müssen, um ein Geschäftsprozessmodell überhaupt transformieren zu können sowie ein Algorithmus, der angewendet werden kann um ein anhand der „Integration Engineering“ Methode erstelltes Geschäftsprozessmodell in ein technisches Geschäftsprozessmodell zu überführen. Weiterhin sollen die Erweiterung der Methodik sowie der erstellte Algorithmus zur Transformation anhand einer prototypistischen Implementierung des Beispielszenarios aus dem E-Government Umfeld evaluiert werden.

3 Stand der Technik

In diesem Kapitel wird ein Überblick über die für die Realisierung relevanten Standards und Technologien gegeben. Abschnitt 3.1 beschäftigt sich mit dem Thema Geschäftsprozessmodellierung und gibt eine Einführung in diese Thematik.

3.1 Geschäftsprozessmodellierung

Mit dem wachsenden Druck eines globalen Marktes ist es für Unternehmungen jeglicher Art von zentraler Bedeutung, sich auf neue, veränderte Marktbedingungen möglichst schnell einzustellen. Dies ist jedoch nur möglich, wenn überhaupt klar ist, wie in einem Unternehmen die einzelnen Glieder in der Wertschöpfungskette zusammenspielen. Es mag sehr simpel klingen, und für Außenstehende scheint diese Frage auch meist einfach zu beantworten, aber betriebsintern sind die realen Aktivitäten meist nur auszugsweise bekannt. So weiß vielleicht eine Sachbearbeiterin in einer Versicherung, wohin sie ein Kündigungsschreiben oder eine Schadensmeldung direkt weitergeben und welche Vorarbeit sie leisten muss. Wie genau mit diesen Dokumenten dann weiter verfahren wird, bleibt ihr verborgen.

Nun könnte man vermuten, dass ein Vorgesetzter genau weiß, wie diese Prozesse ablaufen. Oftmals sind die Aufgaben, die diese Personen übernehmen, jedoch sehr verschieden von denen der Angestellten und der Überblick über den Gesamtprozess wiederum nur auszugsweise möglich.

Daher beobachtet man seit längerem eine Bewegung im betriebswirtschaftlichen Umfeld, die darauf fokussiert, Abläufe im eigenen Wertschöpfungsprozess zu dokumentieren und damit die Weichen zu stellen, diese Prozesse noch effizienter zu gestalten. Dementsprechend wurde das sogennannte „Business Process Reengenieering“ (BPR) und die „Business Process Optimization“ (BPO) zu zunehmend wichtigeren Themen. Beim „Business Process Reengenieering“ wird der bestehende Geschäftsprozess fundamental überarbeitet, um effizienter und effektiver abzulaufen. Dabei kommt es oftmals vor, dass der neu erstellte Geschäftsprozess sich sehr stark vom alten Geschäftsprozess unterscheidet. Die folgende formale Definition findet sich in dem Buch von Hammer und Champy [HC93].

Definition ***Business Reengineering :***
The fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed.

Die „Business Process Optimization“ hingegen untersucht den bestehenden Geschäftsprozess, um kleinere Teilprozesse zu optimieren, ohne fundamentale Änderungen am Prozess vorzunehmen.

S. Weerawarana beschreibt in seinem Buch [WCL+05] ein weiteres Phänomen. Er nennt es die „Virtualisierung von Unternehmen“ oder „virtual enterprises“. Ein Unternehmen, das

heutzutage dem Kunden ein Produkt anbietet, fertigt dieses kaum noch komplett innerhalb der eigenen Betriebsgrenzen. Durch den Kostendruck, den eine globalisierte Welt auf ein Unternehmen ausübt, sind die Betriebe meist dazu gezwungen, einzelne Produktionsschritte auf spezialisiertere und dementsprechend günstigere Unternehmen auszulagern. Dies nennt sich „outsourcing“. Speziell bei Dienstleistungen ist diese Virtualisierung sehr stark zu beobachten.

In vielen Unternehmen kann man sogar sagen, dass das Produkt ein und dasselbe ist, wie der Prozess, durch den es bereitgestellt wird. Ein Beispiel hierfür wären Versicherungen oder Kreditinstitute. Um die Kosten für ein Produkt zu senken, ist es daher essentiell, die Zeit zu minimieren, die es benötigt, um das Produkt auf den Markt zu bringen (sog. time to market).

An dieser Stelle ist es nötig, eine feste Definition von Geschäftsprozessen und Workflows festzuhalten, um mögliche Verwechslungen und Unklarheiten zu vermeiden. Gustavo Alonso definiert in seinem Buch [ACKM04] den Terminus „Geschäftsprozess“ wie folgt:

Definition ***Geschäftsprozess :***
Eine Ansammlung von Aktivitäten, die von menschlichen Benutzern oder Anwendungssystemen [in einer bestimmten Reihenfolge] ausgeführt werden und die nötigen Schritte darstellen, um ein bestimmtes Geschäftsziel zu erreichen.

Weiterhin muss auch der mit dem Geschäftsprozess sehr eng verbundene Begriff Workflow definiert werden.

Definition ***Workflow :***
Formale, auf Computersystemen ausführbare Definition eines Geschäftsprozesses.

Um Prozesse zu beschreiben und grafisch darzustellen, gibt es viele Möglichkeiten. Zu den wichtigsten zählen:

1. Die ARIS Methodik und deren ereignisgesteuerte Prozessketten (EPKs), die im Abschnitt 3.1.1 näher erläutert werden.
2. Die Integration Engineering Methode, die in Abschnitt 3.1.2 beschrieben wird.
3. Petrinetze - auf die in Abschnitt 3.1.3 kurz eingegangen wird.
4. Aktivitätsdiagramme - die in Abschnitt 3.1.4 kurz beschrieben werden.

3.1.1 ARIS Methodik

Eine Methodik, die sich in der betriebswirtschaftlichen Welt etabliert hat, um vorhandene Geschäftsprozesse zu modellieren und zu dokumentieren, ist die ARIS Methodik. Die Methodik für die „Architektur integrierter Informationssysteme“ wurde 1995 von A.W. Scheer und M. Nüttgens in [SNZ95] vorgestellt. Darin wird die Gesamtsicht auf ein Informationssystem in vier Teilsichten aufgeteilt, um die Komplexität des Gesamtsystems möglichst gering zu halten. Die vier Teilsichten werden als ARIS „house of business engineering„ (HOBE) bezeichnet. Sie sind in **Abbildung 5** dargestellt. In der Organi-

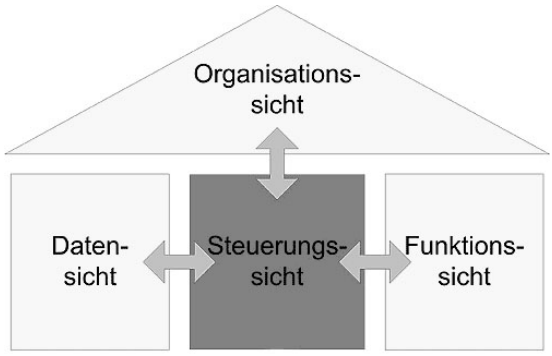


Abbildung 5: Das ARIS house of business engineering [SNZ98]

sationssicht wird die Struktur der Organisation anhand von Organigrammen dargestellt. Zusätzlich werden die verschiedenen Rollen, die ein Person einnehmen kann, definiert. Die Funktionssicht beschreibt die wertschöpfenden Funktionen einer Unternehmung. In der Datensicht werden die verwendeten Datentypen spezifiziert, und die Steuerungssicht beschreibt die wertschöpfenden Prozesse. Jede dieser vier Teilsichten wird wiederum in drei verschiedene Ebenen unterteilt. Diese werden als Fachkonzept, DV-Konzept und Implementierung bezeichnet. Im Fachkonzept werden aus fachlicher Sicht Prozesse abgebildet. Dies wird in der Steuerungssicht anhand von ereignisgesteuerten Prozessketten durchgeführt, auf die in 3.1.1.1 eingegangen wird. Auf der Ebene des DV-Konzepts werden beispielsweise Styleguides und Klassen modelliert, die die Umsetzung des Fachkonzepts in ein Informationssystem betreffen. Die Implementierungsebene stellt dann die Umsetzung des DV Konzeptes in ein lauffähiges System dar.
Die ARIS Methodik wird von dem so genannten ARIS-Toolset unterstützt. Dieses Toolset ist eine Werkzeugplattform, mit deren Hilfe die in der Methodik vorgeschlagenen Diagrammtypen wie beispielsweise Organigramme, EPKs und FZDs erstellt werden können. Dieses Toolset wird von der IDS Scheer AG entwickelt und vertrieben. Die im Rahmen dieser Arbeit tatsächlich verwendeten Diagrammtypen werden nun kurz beschrieben.

3.1.1.1 Ereignisgesteuerte Prozessketten Ereignisgesteuerte Prozessketten sind ein in der Betriebswirtschaft weitverbreitetes Werkzeug, um Arbeitsabläufe zu modellieren. Im ARIS HOBE werden sie auf der Ebene des Fachkonzepts der Steuerungssicht angewandt. Sie wurden von dem Institut für Wirtschaftsinformatik der Univerität des Saarlandes (IWi) in Zusammenarbeit mit der SAP AG entwickelt [KNS92]. Die Methode beruht auf Ansätzen des stochastischen Netzplanverfahrens und Petrinetzen. Der Grundgedanke beruht dabei auf der in der Informatik für Kontrollfluss benutzten und bekannten ECA-Regel (Event Condition Action) [DG96]. Dabei kennzeichnet ein Event einen Zeitpunkt/ein Ereignis und die Bedingung (Condition) den Fall, für den das Ereignis interessant wird. Wenn nun Event und Condition zusammentreffen (z.B. 18:00h und 29C°), wird die spezifizierte Aktion ausgeführt. In den ereignisgesteuerten Prozessketten des ARIS Geschäftsprozessmodells werden nur Ereignisse definiert, die durch Geschäftsfunktionen oder manuell durch externe Akteure angestrengt werden. Dabei werden dementsprechend dann nur interessierende Ereignisse modelliert. Dabei verkürzt sich demnach die ECA Regel auf eine EA Regel, die den EPKs zu Grunde liegt[SNZ98]. Ein Beispiel für eine EPK ist in **Abbildung 6** zu sehen.

Viele Betriebe, die im Zuge einer Business Process Reengineering (BPR) Anstrengung, ihre innerbetrieblichen Abläufe optimieren wollen, greifen auf diese Art der Notation zurück. Betriebswirtschaftler als die Entscheidungsträger in den Führungsetagen der meisten Unternehmen, schätzen diese semi-formale und intuitive Art der Beschreibung von Prozessen. EPKs sind zudem ein gern eingesetztes Mittel zum SAP R3 customizing. Wie oben beschrieben, bilden Ereignisse und Funktionen den Kern der ereignisgesteuerten Prozessketten. Dabei beginnt jedes EPK Diagramm damit, dass ein Ereignis eine Funktion auslöst. Somit stellen Ereignisse „Zeitpunkte“ dar, während Funktionen für ein „zeitverbrauchendes“ Element stehen. Verbunden werden die beiden Elemente anhand von Konnektoren. Es gibt dabei drei verschiedene logische Operatoren, die mit den Konnektoren „eingesetzt“ werden können. Diese sind im Folgenden aufgelistet und beschrieben.

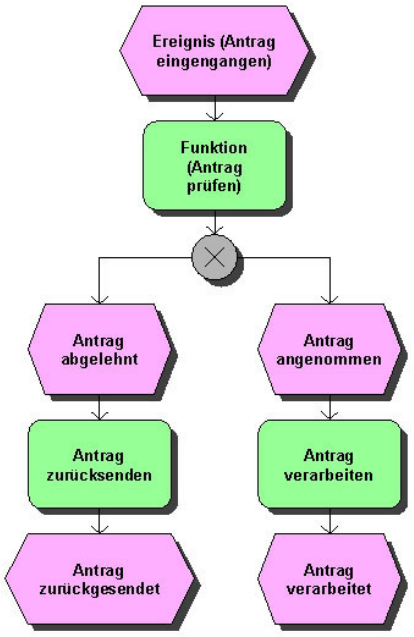


Abbildung 6: Beispiel einer EPK

- Disjunktion (OR) (∨)- kann nur nach Funktionen auftreten, da nur Funktionen „entscheiden“ können (Entscheidungsfunktionen). Logisch bedeutet das OR, dass mindestens einer der ausgehenden Pfade „begangen“ werden muss.
- Antivalenz (XOR) (X) - auch dieser Operator kann nur nach einer Funktion vorkommen. Es darf nur genau einer der ausgehenden Pfade „begangen“ werden.
- Konjunktion (AND) (∧) - Dies ist das einzige Konstrukt, das auch nach Ereignissen

auftreten darf, da hierbei alle Pfade zu „begehen“ sind und somit keine Entscheidung notwendig ist.

Heinrich Seidlmeier fasst die Regeln, die bei der Modellierung gelten in seinem Buch [SNZ02] wie folgt zusammen.

- Jede EPK beginnt und endet mit einem Ereignis
- Ereignisse und Funktionen wechseln sich (in der Regel) ab
- Aus und in Funktionen läuft nur jeweils eine Kontrollflusskante
- Kein Objekt steht ohne Kante im Modell
- Eine Kante verbindet (in der Regel) genau zwei verschiedene Objekte
- Nach einem Ereignis darf kein ODER bzw. XOR-Konnektor stehen
- Durch Konnektoren verzweigte Teilabläufe (Pfade) werden durch gleichartige Konnektoren wieder zusammengeführt
- Werden mehrere Pfade mit einem Konnektor wieder verbunden, darf der Konnektor nur eine auslaufende Kante besitzen
- Direktverbindungen von Konnektoren sind erlaubt

In einem „Funktions-Zuordnungsdiagramm“ werden beteiligte Personen und IT-Systeme modelliert, die eine Funktion realisieren. Dabei ist die Granularität der Beschreibung der Integrationsrelevanten Aspekte, wie bereits erwähnt, auf die Anwendungssystem Ebene beschränkt.

3.1.2 Integration Engineering Methodik

Die „Integration Engineering“ (IE) Methodik wurde in Anlehnung an die ARIS Methodik im Rahmen des Forschungsprojektes Integration Engineering [FTH05] entwickelt. Dabei wird im Gegensatz zur ARIS Methodik darauf verzichtet, EPK Diagramme mit weiteren Details zu „überfrachten“. So werden, wie bei der ARIS Methodik, Organisationseinheiten und Entitätstypen auf so genannte „erweiterte Funktionszuordnungsdiagramme“ ausgelagert. Diese werden in Abschnitt 3.1.2.1 beschrieben. Es wird weiterhin darauf verzichtet, so genannte Funktionswegweiser zu verwenden. Prozesse als ganzes können nur als Prozessmodul dargestellt werden. Zusätzlich zu den in der ARIS Methodik verwendeten Symboltypen, können Funktionen auch noch mit Anforderungen versehen werden. Diese können auch in einer Prozessübersicht an das Prozessmodul geknüpft werden. Die technische Detaillierung wird auf die folgend beschriebenen eFZDs ausgelagert.

3.1.2.1 Erweiterte Funktionszuordnungsdiagramme (eFZDs) Die so genannten erweiterten Funktionszuordnungsdiagramme sind eine, wie der Name schon verrät, Erweiterung der dem Objekttyp Funktion hinterlegten Funktionszuordnungsdiagramme. Sie wurden im Rahmen des Integration Engineering Forschungsprojektes erarbeitet und werden benutzt, um die Komplexität von EPKs niedrig, sowie die Übersicht hoch zu halten. Dabei wird in eFZDs der Datenfluß von Input- zu Outputdaten mit den beteiligten Systemen, Operationen und Entitätstypen dargestellt. Dabei verfeinern sie die in der ARIS Methodik verwendete Notation um technische Details wie die von einem Anwendungssystem verwendeten Methoden (Operationen) und Entitäten die zwischen den Operationen ausgetauscht werden. Damit bilden sie die vorherrschende Applikationslandschaft sehr viel detaillierter ab als es in der ARIS Methodik vorgesehen ist.

Folgende Objekttypen sind in eFZDs erlaubt.

- Funktionen
- Operationen, mit Attributen Web Service, GUI, Systemintern
- Kanten - (unterstützt, ist Input für, hat Output, führt aus)
- Anwendungssysteme
- Aktivitäten
- Entitätstypen
- Personentypen
- Notizen

Funktionen stellen die detaillierter dargestellte Funktion dar, werden aber der Übersicht halber meist weggelassen. Zentraler Aspekt der eFZDs sind die `Anwendungssysteme`, die in einer Art Repository erstellt werden, und deren Ausprägungen in den eFZDs mit den durch sie zur Verfügung gestellten `Operationen` versehen werden. Dies geschieht durch eine Kante des Typs „*unterstützt*“.

Operationen können verschiedene Attribute haben, die ihre technische Realisierung und Zielsetzung verfeinern sollen. Es gibt drei verschiedene vordefinierte Attribute, aus denen gewählt werden kann. Einerseits den Typ *Web Service* der indiziert, dass die vorhandene Operation durch eine *Web Service* Schnittstelle bereitgestellt wird, andererseits *GUI*, welches anzeigt, dass es sich um ein Benutzerinterface handelt sowie *Systemintern*, um Verarbeitungsschritte innerhalb des Systems anzuzeigen.

Der Output (kante „hat Output“) beziehungsweise Input (Kante „ist Input für“) einer Operation ist immer ein `Entitätstyp`, der durch eine gerichtete Kante zugeordnet wird.

Personentypen interagieren, modelliert durch eine Verbindung/Kante („führt aus“) von **Aktivität** und `Personentyp` mit dem `Anwendungssystem` und deren angebotenen `Operationen`. Dabei werden wie in **Abbildung 7** zu sehen ist, Daten (**Entitätstypen**)

von den Aktivitäten „geändert“. Das in **Abbildung 7** dargestellte eFZD verdeutlicht die textuell beschriebenen Regeln. Die betrachtete Darstellung wird nachher im Szenario noch verwendet.

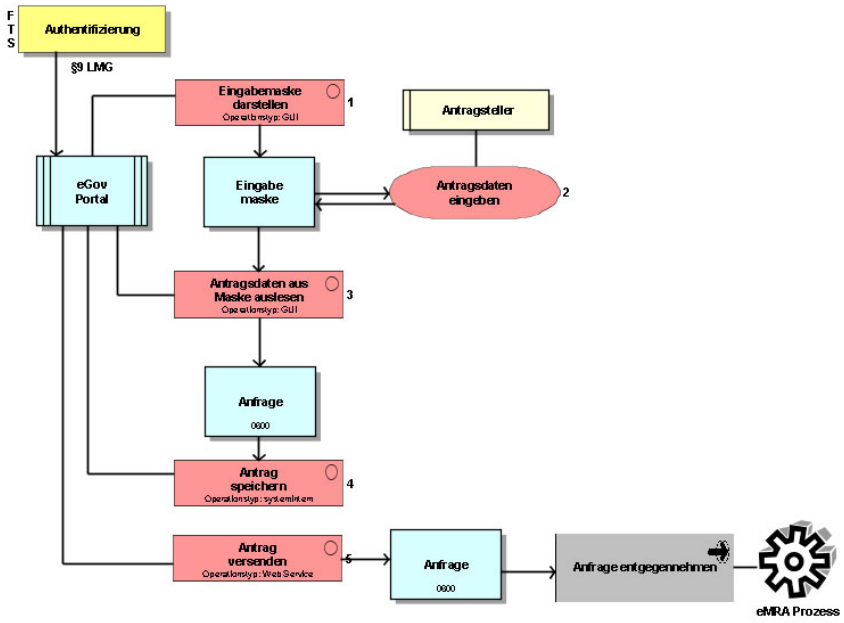


Abbildung 7: Beispiel eines eFZD

3.1.3 Petrinetze

Petrinetze wurden in den 1960ern von Carl Adam Petri erfunden [Pet62] und sind durch die intensive Forschungsarbeit, die sie über die Jahre erfahren haben, ein sehr reifes Notationsmodell. Ein Petrinetz ist ein bipartiter, gerichteter Graph. Er besitzt Plätze (Knoten) und Transitionen (gerichtete, gewichtete Kanten). **Abbildung 8** zeigt ein typisches Petrinetz. Der eigentliche Wert für Petrinetze in Bezug auf die Geschäftsprozessmodellierung besteht darin, dass es viele Tools gibt, die den Benutzer bei der Erstellung und Validierung von Petrinetzen im syntaktischen Sinne unterstützen. Es gibt sogar einige Werkzeuge, die sich auf die Beschreibung von Geschäftsprozessen anhand von Petrinetzen „spezialisiert“ haben (Promatis Process Designer, Woflan, WoPeD [FFG+04]). Da sich diese Art der Notation jedoch in den Führungsetagen der Unternehmen kaum durchgesetzt hat, weil sie den Entscheidungsträgern zu formal ist, spielt sie eine untergeordnete Rolle.

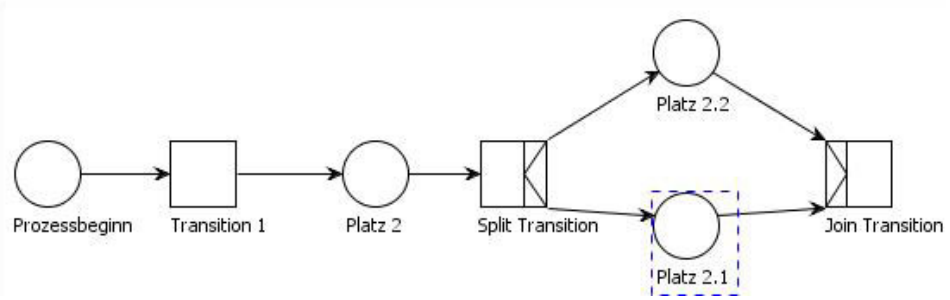


Abbildung 8: Petrinetz Beispiel

3.1.4 Aktivitätsdiagramme

Aktivitätsdiagramme sind ein Diagrammtyp, der in der Unified Modelling Language (UML) definiert wurde. Ziel von Aktivitätsdiagrammen ist eine genaue Beschreibung von Abläufen und Beziehungen zwischen einzelnen Aktionen/Zuständen in einem Computersystem. **Abbildung 9** zeigt ein typisches Aktivitätsdiagramm. Jedes Aktivitätsdiagramm hat einen Startzustand (gefüllter Kreis links) und einen Endzustand (Doppelkreis ganz rechts). Zustände sind als abgerundete Rechtecke dargestellt und mit einer Beschriftung markiert. Um von einem Zustand in den nächsten zu gelangen, muss eine gerichtete Kante (Transition) in die gewünschte Richtung bestehen. Diese dienen dazu, den Kontrollfluss in Systemen zu beschreiben.Um zwischen verschiedenen Nutzergruppen oder Systemen zu unterscheiden, kann das Aktivitätsdiagramm in so genannte „swimlanes“ unterteilt werden.
Es existieren zwei Versionen der Aktivitätsdiagramme (Activity Diagrams). Einerseits die der UML 1.x, andererseits die der Version 2.0. In UML 2.0 ist es zusätzlich zur oben beschriebenen Funktionalität noch möglich, Objektfluss bzw. Datenfluss zu modellieren. Dazu kann an die Zustände ein Ein-/Ausgabepin angebracht werden (kleines Rechteck an rechter/linker Seite des Zustandes). Zusätzlich können geschachtelte Modelle verwendet werden und Objekte hinterlegt werden. Mit der Version 2.0 der Unified Modelling Language überwindet diese Art von Diagrammtyp immer mehr seine „Kinderkrankheiten“. Die Diagramme sind aussagekräftiger geworden und eignen sich nun besser als in Version 1.x zum Modellieren auch komplexer Geschäftsprozesse. Im Moment ist jedoch kein Werkzeug am Markt, das sich der speziellen Belange der Geschäftsprozessmodellierung annimmt. Im Gegensatz zu EPK's sind Aktivitätsdiagramme im betriebswirtschaftlichen Umfeld noch nicht etabliert [KFG02].

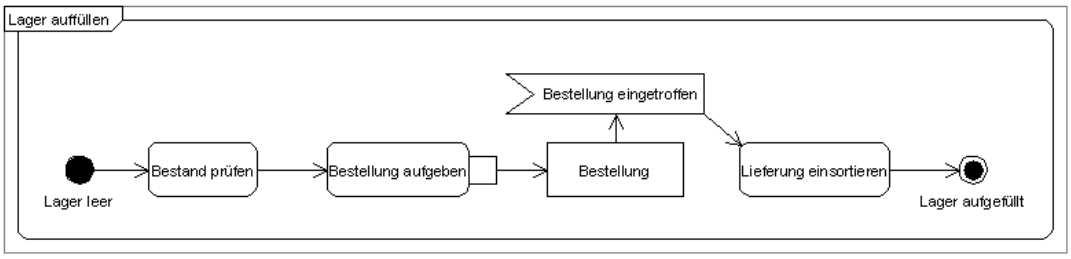


Abbildung 9: Beispiel Lagerauffüllen (UML 2.0)

3.2 Service-orientierte Architektur

Die in Abschnitt 3.1 beschriebenen Einflussfaktoren wirken sich natürlich auch auf die in der Informations- und Kommunikationstechnik (IT) vorherrschenden Architekturen aus. Die von der betriebswirtschaflichen Seite aus geforderte Flexibilität über Plattformgrenzen und Unternehmensgrenzen hinweg, forderte nach einer weiteren „Virtualisierung“ von Anwendungsdiensten. Um die Forderungen nach Flexibilität zu realisieren, bedient sich die SOA zweier, in der Softwareentwicklung länger bekannten Architekturparadigmen.

1. Loose coupling - geringe Kopplung zwischen den einzelnen Komponenten einer Service-orientierten Architektur
2. Design by contract [Mey88] - sog. Vertragsbasierte Entwicklung

Loose coupling ist ein Schlagwort, das seit den 60er Jahren im Softwareengineering gebraucht wird [Tan95]. Es bezeichnet den losen Zusammenhalt von Teilkomponenten einer Gesamtanwendung. Loose coupling ist deswegen sehr erstrebenswert, da einzelne Teile einer Anwendung, wenn sie nur lose gekoppelt sind, leichter durch beispielsweise aktualisierte/erneuerte Komponenten ersetzt werden können, ohne Nebenwirkungen auf andere Systemteile zu haben. Je weniger Abhängigkeiten ein Modul zu einem anderen Modul hat, desto loser ist es gekoppelt. Man könnte auch sagen, je weniger Annahmen ein Modul über ein anderes Modul macht, desto loser ist es gekoppelt. Es sollte jedoch klar sein, dass die Vorteile, die diese Art von loser Kopplung hat, auch ihre Nachteile mit sich bringt. So ist leicht nachzuvollziehen, dass zwei Module, die sehr lose gekoppelt sind, nie die Performanz haben werden, wie zwei Module die gleiche Funktionalität, jedoch hohe Kopplung haben.

Design by contract wurde erstmals von Bertrand Meyer [Mey88], Mitarbeiter von Interactive Software Engineering und Erfinder der Sprache Eiffel, formuliert. Dieses Paradigma geht davon aus, dass zwischen 2 Komponenten feste „Verträge“ geschlossen werden. Dabei werden Vorbedingungen und Nachbedingungen „formuliert“, die erfüllt sein müssen, wenn der Vertrag eingehalten wurde. Falls eine Vorbedingung nicht erfüllt wurde, wird ein Fehler „geworfen“.

Diese zwei Paradigmen sollen die Flexibilität der Architektur garantieren. Das OASIS Komitee [OAS], welches für die Standardisierung der Referenzarchitektur zuständig ist, definiert in ihrem Referenz Modell für Service Orienientierte Architektur [MLM+06] unter

anderem folgende wichtige Begriffe.

Definition ***Service :***
The means by which the needs of a consumer are brought together with the capabilities of a provider

Definition ***Service Provider :***
An entity (person or organization) that offers the use of capabilities by means of a service.

Definition ***Service Consumer :***
An entity which seeks to satisfy a particular need through the use capabilities offered by means of a service.

Service übersetzt bedeutet nichts anderes als Dienstleistung. Und wie in der Realität gibt es außer den beiden bereits erwähnten „Rollen " noch die Rolle des Vermittlers. Diese sind in **Abbildung 10** zu sehen und werden im Folgenden kurz beschrieben.

- Einerseits gibt es einen „Kunden", der eine Dienstleistung in Anspruch nehmen will. In dieser Arbeit wird er Dienstleistungs-Konsument genannt (**vgl. Service Requestor**/*Sevice Consumer*).

- Um die Dienstleistung, die der Kunde in Anspruch nehmen will zu finden, braucht man eine Stelle, an der sich der Konsument informieren kann, welche Dienstleistungen am Markt angeboten werden. Diese sogenannte Auskunft (**vgl. Discovery Service**) ist zu verstehen wie ein Dienstverzeichnis. Jede neue Dienstleistung, die erstellt wird, sollte seine „Kontaktinformationen" in diesem Register ablegen.

- Andererseits sollte es natürlich auch die Art von Dienstleistung bzw. den Dienstleister geben, der die von dem Konsument geforderte Dienstleistung umsetzt. Dieser wird Dienstleister genannt (**vgl. Service Provider**).

Das Zusammenspiel dieser drei Rollen in der Service-orientierten Architektur kann sehr eindrucksvoll an dem sogenannten SOA Dreieck (SOA Triangle, **Abbildung 10**) verdeutlicht werden. Nach der Erstellung eines Dienstes, legt er seine „Kontaktdaten" (vgl. 1. Publish) in der Service Discovery ab. Braucht ein anderer Dienst diese Funktionalität, durchsucht er die Service Discovery (vgl. 2. Discover/Find) und liest die Kontaktdaten aus, um im nächsten Schritt (vgl. 3. Bind) den gefundenen Dienst zu kontaktieren.

Oft wird SOA in einem Zug mit Web Services genannt. Dabei scheint die Meinung vorzuherrschen, eine Service-orientierte Architektur sei nur mit Hilfe von Web Service zu

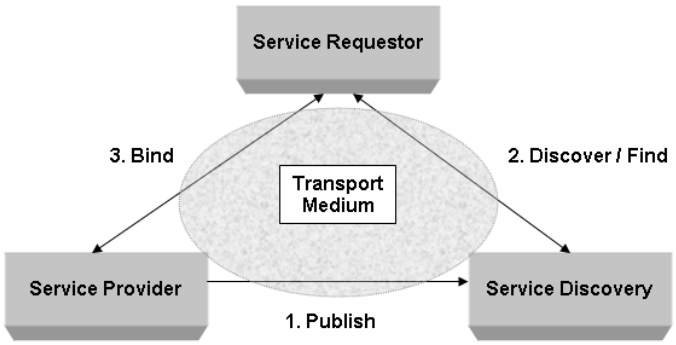
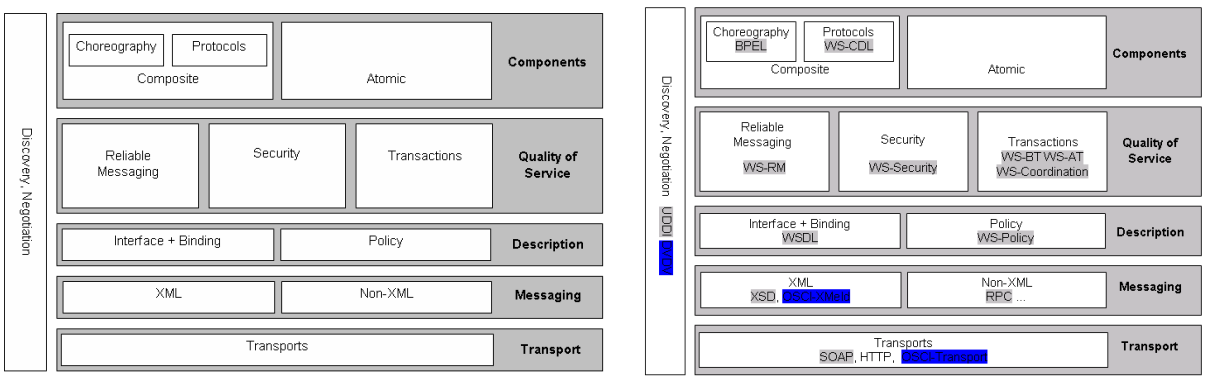


Abbildung 10: Das SOA Dreieck nach [WCL+05]

realisieren. Dies ist schlichtweg falsch. Web Services sind nur ein Bausteintyp, der zum Design einer SOA genutzt werden kann. Es gibt jedoch durchaus andere Möglichkeiten, eine Service-orientierte Architektur zu realisieren. So sind beispielsweise die DCOM Architektur von Microsoft oder die CORBA Architektur sowie J2EE mögliche Implementierungen einer SOA.

In **Abbildung 11a** sind die wichtigsten Komponenten einer Service-orientierten Architektur zu sehen. In dieser Arbeit wird diese allgemein gehaltene Abbildung in Bezug auf die in der Realisierung verwendete Web Service Implementierung noch näher beschrieben. Diese um die jeweils definierten Web Service Standards erweiterte Abbildung ist in **Abbildung 11b** zu finden.



(a) (b)

Abbildung 11: a) Der SOA Stack [WCL+05] b) spezialisiert auf WS Standards

Ein wichtiger Aspekt, der bei der Anwendungsintegration bedacht werden sollte, ist die Zukunftsfähigkeit der eingesetzten Technologien. Service-orientierte Architektur , die

durch Web Services realisiert ist, wird durch den Einsatz frei verfügbarer, am Markt etablierter Standards, dieser Anforderung gerecht. Die größten Hersteller von EAI Lösungen haben sich der Unterstützung und der Einhaltung dieser Standards verschrieben. So wirkten an fast allen, in den folgenden Abschnitten vorgestellten Standards Firmen wie Microsoft, IBM, SAP, Siebel, BEA und Oracle mit. Damit ist eine gewisse Sicherheit bei der Portierung von Lösungen, die anhand dieser Standards erstellt worden sind, gewährleistet. In **Abbildung 11b** wird der in **Abbildung 11a** vorgestellte, so genannte SOA Stack, um die korrespondierenden WS Spezifikationen erweitert (dunkelgrau hinterlegt). Auch die im E-Government relevanten Spezifikationen wurden zugeordnet (hellgrau hinterlegt).

Die in diesem Kapitel nicht behandelten Standards sind WS-RM (reliable messaging), WS-Policy und WS-CDL (coreography description language). WS-RM ist eine Spezifikation zur Unterstützung von „reliable messaging“. Durch sie wird ein Framework zur Verfügung gestellt, das es beispielsweise ermöglicht, die Reihenfolge von ausgetauschten Nachrichten (Messages) zu erhalten, oder die Vollständigkeit von Nachrichtensequenzen zu gewährleisten.

Die WS-Policy Spezifikation stellt ein wichtiges Mittel zur Verfügung, um geforderte oder gebotene „quality of service“ (Dienstqualität) zu beschreiben. Dies hat besonders hohe Bedeutung, wenn über Unternehmensgrenzen hinweg Web Services angeboten oder genutzt werden.

Die WS coreography description language (WS-CDL) ist eine XML-Notation, die es ermöglichen soll, das „von außen“ sichtbare Verhalten von einzelnen WS Komponenten, in einem Peer to Peer Gesamtprozess zu beschreiben. Im Gegensatz zu BPEL, das einen Prozess anhand des Kontroll- und Datenflusses prozesszentrisch beschreibt, werden in der WS-CDL die Reihenfolge der Nachrichten zwischen den Partnern beschrieben. Ein denkbares Szenario für den Einsatz von WS-CDL wäre unter Umständen die Verifikation der Kompatibilität verschiedener BPEL Prozesse, die innerhalb eines Gesamtprozesses miteinander kommunizieren.

3.2.1 Web Services

Ein mögliches Design für eine Service-orientierte Architektur stellen Web Services dar. Web Services versuchen die Problematik der Anwendungsintegration durch einen sehr simplen, aber bekannten Ansatz zu lösen. Probleme bei einer Anwendungsintegration entstehen immer dann, wenn Implementationen, die verschiedene Programmierparadigmen benutzen, zu viele Annahmen über den „Partner“ anstellen. Das Fundament von Web Services bilden die zwei folgenden Aspekte: Web Services stellen ihren Partnern eine wohldefinierte Schnittstelle zur Verfügung. Diese Schnittstelle beschreibt alle für eine Interaktion notwendigen Gegebenheiten. Der zweite Aspekt ist, dass diese Schnittstelle die Implementierungsdetails vor seinen Partnern „versteckt“. Es ist demnach irrelevant, ob der Web Service in Form einer Java Anwendung realisiert ist, oder mit dem .NET

Framework implementiert wurde. Diese Form der Abstraktion bringt für die Komposition von Web Services viele Vorteile, da Komponenten, die keine Annahmen über ihre Partner machen, einfacher zu Kompositionen zusammenzufügen sind. Man kann jedoch davon ausgehen, dass die Performanz solcher Kompositionen, im Vergleich zu Kompositionen, die beispielsweise durch RPC/RMI integriert wurden, eher hinten anstehen.

Um die oben genannten Web Service Schnittstellen zu beschreiben, wurde die sogenannte Web Service Description Language [CCMW01] definiert, die im Folgenden näher erläutert wird.

3.2.2 WSDL 1.1

Die von dem OASIS [OAS] Komitee, in der Version 1.1 [CCMW01] spezifizierte Web Service Description Language (WSDL) ist, wie der Name schon verrät eine Schnittstellenbeschreibungssprache. Man könnte sie mit der aus der CORBA Welt bekannten Interface Definition Language (IDL) vergleichen. In einem WSDL Dokument werden alle wichtigen Schnittstellen eines Web Services beschrieben. Ein WSDL Dokument besteht grundsätzlich aus zwei Teilen, einem abstrakten und einem konkreten. Der abstrakte Teil des WSDL Dokuments (s. **Abbildung 12** grau hinterlegt)enthält die Konstrukte `Types`, `Messages` und `PortTypes`. In dem Abschnitt `Types` werden die von dem Service benutzten Typendeklarationen vorgehalten. Diese sind maschinen- und sprachneutral. Sie werden meistens anhand der XML-Schema „Notation“ beschrieben. `Messages` sind Kontainer für die von einer Funktion verwendeten „Parameter“, die nach input und output getrennt sind. So gibt es für jeden Input sowie für jeden output eine Nachricht. In der `PortTypes` Deklaration werden einzelne Teile von Messages/Nachrichten einzelnen Operationen zugeordnet.

Im konkreten Teil (s. **Abbildung 12** weiss hinterlegt) einer WSDL 1.1 Spezifikation wird anhand der Konstrukte `Bindings` und `Services` die konkrete Realisierung des Web Services spezifiziert.

In dem folgenden Abschnitt werden die Konstrukte der Version 1.1 näher beschrieben.

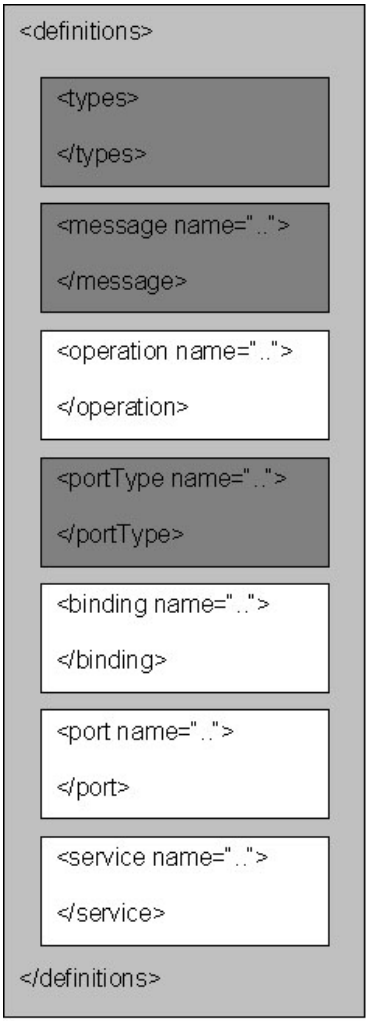


Abbildung 12: Aufbau der WSDL 1.1

- `<types>` - In diesem XML-Konstrukt werden die Typendefinitionen der verwendeten Datenstrukturen beschrieben. Es ist möglich, aus schon vorhandenen XML-Schema Dateien Datenformate zu importieren. Anhand der durch XML-Schema beschriebenen Typen können dann die im nächsten Punkt beschriebenen Messages „komponiert“ werden. Die beschriebenen Typendefinitionen sind maschinen- und sprachneutral.
- `<messages>` - Messages sind einzelne Nachrichten, die aus den zuvor definierten Types zusammengestellt werden. Eine Message kann über mehrere Messageparts verfügen. Messages können als Funktionsparameter gesehen werden. Es gibt jeweils eine Message für jeden Out- bzw. Input.
- `<operation>` - Dieses Konstrukt dient dazu, die einzelnen Operationen, die ein Web Service zur Verfügung stellt, zu beschreiben.
- `<PortType>` - Ein PortType beschreibt eine abstrakte Funktionalität, die ein Web Service zur Verfügung stellt, ohne auf seine konkrete Implementierung einzugehen. Ein Porttype, in Verbindung mit einem binding, ergibt einen konkret ansprechbaren Port.
- `<binding>` - Das Binding stellt die konkrete Bindung der Kommunikation an ein Transportmedium dar. Das bedeutet, dass ein PortType mit einem speziellen, beispielsweise HTTP binding, einen reellen Port „implementiert“
- `<port>` - Ein Port stellt einen konkret adressierbaren Endpunkt(Endpoint) dar. Jeder Port hat einen PortType, der wiederum verschiedene Operationen aggregiert. Ein Port ist immer an ein bestimmtes binding gebunden.
- `<service>` - Ein Service kann als Container für Ports gesehen werden, die im Rahmen eines Services angeboten werden. So stellt beispielsweise der Service „Kontoführung“ unter Umständen die Ports „httpÜberweisung“, „httpKontostandAbfragen“ und „httpKontoAuflösen“ zur Verfügung.

3.2.3 SOAP

Das SOAP Protokoll (ehem. Simple Object Access Protokoll genannt)[WCL+05] stellt eine mögliche, standardisierte Variante dar, wie Web Services miteinander kommunizieren können. SOAP ist nicht an ein bestimmtes Transportmedium gebunden, sondern beschreibt lediglich eine standardisierte Nachrichtenarchitektur/struktur. Die Nachrichten werden dabei in eine oder mehrere Kopfzeile(n) („header“) und genau einen Nachrichtenrumpf („body“) gegliedert. Da a priori keine Aussage über den „Pfad“ einer Nachricht getroffen werden kann, definiert SOAP drei verschiedene Rollen entlang des Nachrichtenpfades vom Sender über den Intermediär („next“) bis hin zum Empfänger („ultimate-Receiver“). Zusätzlich zu diesen zwei Rollen wird noch eine „Niemand“ Rolle („none“) definiert, die es erlaubt, Kopfzeilen einzufügen, die Daten beinhalten, deren Verarbeitung nirgends „formal“ erzwungen wird. Dies könnten beispielsweise Daten sein, die andere Kopfzeilen erweitern/präzisieren.

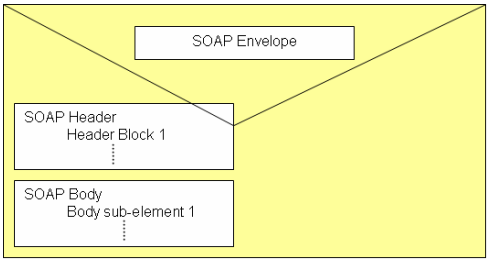


Abbildung 13: SOAP Nachrichtenstruktur

Da sich SOAP nicht auf eine Transportmethode festlegt, ist ein SOAP Nachrichtenaustausch über viele Protokolle möglich. Die am häufigsten verwendete Methode ist SOAP auf HTTP Basis. Es ist aber auch SOAP auf Basis von RPC oder beispielsweise SMTP möglich.

3.2.4 Universal Data Description and Discovery (UDDI)

Ein elementarer Teil einer jeden Integration auf Basis einer Service-orientierten Architektur ist ein Discoveryservice für einzelne Komponenten. Im Falle einer SOA, die in Form von Web Services realisiert wird, existiert ein Standard namens UDDI (Universal Data Description and Discovery), der vom OASIS Komitee in der Version 3.0.2 im Februar 2005 standardisiert worden ist (siehe [OAS05]).

Eine UDDI Komponente sollte nicht nur als einfaches Telefonbuch für Dienste verstanden werden. Aspekte, die bei der Entwicklung von Lösungen im Service-orientierten Umfeld eine Rolle spielen, sind beispielsweise die Funktion einer UDDI als Dienstverzeichnis. So greifen Entwickler, die eine Anwendung erstellen, auf das Verzeichnis zu, um Komponenten, die für ihr System benutzt werden könnten, zu finden. Gerade in größeren Unternehmen, die über eine eigene Softwareentwicklungsabteilung verfügen, ist diese Funktionalität

sehr wichtig. Das Stichwort hierzu heißt Wiederverwendbarkeit. In diesem Zusammenhang sollte jede UDDI ein einfaches Interface bereitstellen, das das Suchen nach bestimmten Services ermöglicht. Dies kann einerseits ein Webinterface sein, das dem Benutzer die Möglichkeit, gibt nach bestimmten Services zu suchen, andererseits auch eine Standard API, die es dem Entwickler ermöglicht, Anfragen direkt über seine IDE an die UDDI zu senden.

Ein anderer wichtiger Aspekt, ist das dynamische Binden von Web Service Partnern zur Laufzeit. Um dies sinnvoll zu ermöglichen, sind in der Spezifikation des UDDI Dienstes mehrere standardisierte WSDL Interfaces festgelegt worden. Mithilfe dieser Schnittstellen können Anfragen direkt zur Laufzeit genutzt werden, um beispielsweise den „billigsten“ Service zu finden, den es zum Zeitpunkt der Anfrage gibt.

Einer der wichtigsten Punkte, die eine UDDI benutzbar machen, ist die Kategorisierung. Um eine sinnvolle Kategorisierung zu ermöglichen, ist es nötig, ein aussagekräftiges, möglichst generelles Datenmodell zu verwenden. **Abbildung 14** zeigt einen Ausschnitt des Datenmodells, wie er in der Spezifikation 3.0.2 festgelegt ist.

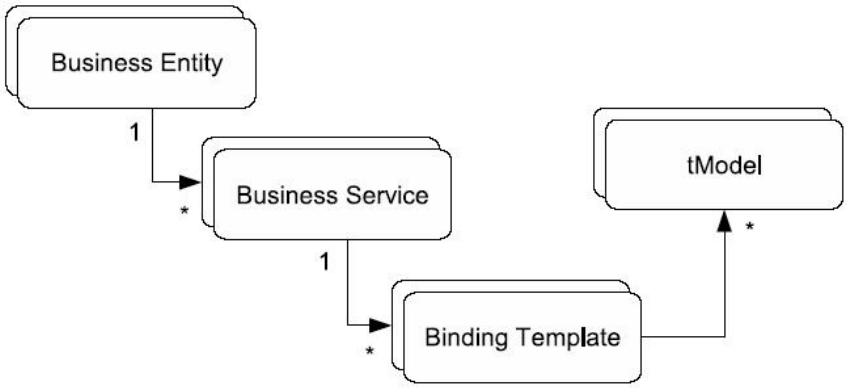


Abbildung 14: Das UDDI Datenmodell nach [OAS05]

3.2.5 WS-Security

Sicherheit ist wohl eines der wichtigsten Aspekte, wenn es darum geht, sich für ein Design einer SOA zu entscheiden. Als SOAP veröffentlicht wurde, war einer der Hauptkritikpunkte das Fehlen einer Sicherheitsinfrastruktur und das Fehlen einer Unterstützung von Transaktionen. Dies ist jedoch größtenteils ungerechtfertigt, da SOAP nur ein Nachrichtenformat beschreibt, nicht jedoch die Kriterien, die den Austausch der Nachrichten betreffen. Bei der Erstellung der WS Standards wurde sehr starker Wert auf Modularität und „separation of concerns“ gelegt. Aspekte, wie das oben genannte transaktionale Verhalten und die Sicherheit, sollten auf der Basis des SOAP Standards zusätzlich separat standardisiert werden.

WS Security [LK06] befasst sich hauptsächlich mit zwei fundamentalen Punkten, wenn

es um Sicherheit geht. Einerseits mit dem Punkt der Nachrichtenintegrität, andererseits mit der Vertraulichkeit der gesendeten Nachrichten. Im WS-Security Standard wird eine Erweiterung der SOAP Nachrichten um ein zusätzliches Headerfeld beschrieben. Dieser Headerblock heißt Security und soll eine Signatur enthalten. In diesem Headerblock werden alle sicherheitsrelevanten Daten abgelegt. WS-Security spezifiziert auch weiterhin, welche Arten von Elementen in diesem Security Header benutzt werden können und wie einzelne Knoten des Nachrichtenpfades die sicherheitsrelevanten Informationen zu verarbeiten haben.

3.2.6 WS-Coordination

Aufbauend auf den WS Spezifikationen WSDL und SOAP bietet WS-Coordination [CFFF05] ein Gerüst um verteilte Interaktionen zwischen WS zu unterstützen. Dabei geht WS-Coordination nicht so weit, spezielle Transaktionsarten zu unterstützen, sondern bietet ein erweiterbares Grundgerüst für einen Koordinationsservice. Kernkonstrukte des Koordinationsservices sind dabei:

- ein Activation Service - bei dem eine Applikation einen so genannten Koordiantionskontext für eine Transaktion beantragen kann
- einen Registration Service - bei dem sich Teilnehmer für eine Interaktion registrieren können
- ein Koordinationstypspezifisches Set von Koordinationsprotokollen

Transaktionen kommen seit langem in Computersystemen vor. Speziell bei sensiblen Daten ist es unabdingbar, dass einige Kriterien eingehalten werden. Diese Kriterien werden auch das ACID Prinzip genannt [Mit06]. Kaum eine Geschäftsbeziehung kann ohne Transaktionen auskommen. Dieser Thematik wurde sich in mehreren WS Spezifikationen angenommen. Dabei wird unterschieden zwischen langlaufenden Transaktionen (siehe Abschnitt 3.2.6.1), die das ACID Prinzip in etwas abgeschwächter Form realisieren, und herkömmlichen Transaktionen von eher kurzer Laufzeit (siehe Abschnitt 3.2.6.2).

3.2.6.1 WS-Business Activity Langlaufende Geschäftsprozesse sind mit dem herkömmlichen ACID Prinzip und den daraus resultierenden Konsequenzen, wie dem Sperren der benutzten Ressourcen, nicht zu realisieren. Daher wurde ein Standard geschaffen, der auf Basis des WS-Transaction Frameworks eine Unterstützung für diese Art von „Transaktionen“ zur Verfügung stellt. Die wichtigsten Charakteristiken von Business Transactions sind laut Spezifikation ([CCF+05b])

- Business Activities (BA) können über einen längeren Zeitraum eine Vielzahl von Ressourcen konsumieren
- es können innerhalb einer BA eine Vielzahl von Atomic Tranactions beinhaltet sein

- Teilergebnisse der Aktivität können vor Abschluss der kompletten BA eingesehen werden
- Die Antwort auf eine Anfrage kann unter Umständen sehr lange dauern
- Falls eine Fehlersituation eintritt, reicht ein Abbruch der BA nicht aus. Unter Umständen muss spezifische Logik abgearbeitet werden, um die Ergebnisse einer Aktion zu widerrufen

3.2.6.2 WS-Atomic Transaction Die WS-Atomic Transaction Spezifikation beruht wie auch der WS-Business Activity Standard auf dem WS-Transaction Framework. Die durch den WS-Atomic Transaction Standard unterstützten Transaktionsarten können einfach charakterisiert werden. Sie stellen alle die Anforderung, dass das ACID Prinzip erfüllt wird. Das bedeutet, dass alle Änderungen, die an Ressourcen durch einen Teilnehmer der Transaktion gemacht werden sollen, so lange nicht endgültig sein dürfen, bis der Koordinator der Transaktion eine „commit“ Nachricht versendet. Falls der Koordinator von einem der Teilnehmer eine Fehlernachricht bekommt, wird die ganze Transaktion rückgängig gemacht (rollback). Jeder Teilnehmer muss in diesem Falle davon absehen, die Änderungen, die ausstanden, durchzuführen. Der Zustand der Gesamtapplikation nach einem Abbruch (abort) muss der gleiche sein wie vor dem Ausführen der abgebrochenen Transaktion. Meist sind diese Art der Transaktionen sehr kurzlebig. Die Spezifikation des Standards ist unter [CCF+05a] zu finden.

3.3 Workflowmanagement Systeme (WfMS)

Die ersten Workflowmanagement Systeme entstanden Anfang bis Mitte der 90er Jahre. Damals wurde nach einem Weg gesucht, wie einfache administrative Aufgaben mithilfe von vorhandenen Computersystemen bestmöglich unterstützt werden könnten. Meistens waren die zu automatisierenden Prozesse papiergestützt. Zielsetzung dieser Systeme war, Fehler durch Medienbrüche zu minimieren. So waren die ersten WfMS auch eher eine Art „Dokumenten routing Maschinen“ (document routing systems). Sie waren dafür verantwortlich, die benötigten Dokumente auf ihrem Weg durch die Administration zu lotsen. Diese Art von Prozessen werden administrative Workflows genannt [GHS95] und werden durch die meisten Dokumenten Management Systeme (DMS) unterstützt.

Mitte bis Ende der 90er Jahre entstand durch die in Kapitel 2 beschriebenen Faktoren eine starke Nachfrage an Enterprise Application Integration (EAI) Produkten. Man erkannte bald, dass WfMS mehr leisten konnten, als nur Dokumente an (menschliche) Benutzer zu routen. Sie sollten nun die Aufgabe übernehmen, die Logik zu implementieren, die nötig war, um größere, heterogene und verteilte Systeme zu integrieren. Diese Art von Workflows, deren Schwerpunkt eher die Integration von verteilten Anwendungssystemen ist, nennt man „production workflow“ [LR99].Doch die Funktionalität früher WfMS reichte nicht aus, um all den komplexen Anforderungen zu genügen, die an EAI Systeme gestellt wurden. So wurden dann auch WfMS mit EAI Lösungen kombiniert, um

den Anforderungen der Benutzer gerecht zu werden. Dabei übernahmen die EAI Teile des WfMS die Aufgabe der „Verdrahtung“ der einzelenen Systeme, während die WfMS die Unterstützung der Geschäftsprozesslogik gewährleistete.

Die in **Abbildung 15** aufgeführten Produkte wurden in dieser Arbeit auf Ihre Benutzbarkeit in Hinsicht auf Geschäftsprozessintegration evaluiert. Die Hersteller dieser Produkte sind die größten „Player“ auf diesem Markt und haben sich in mehreren Konsortien der Standardisierung der benutzten Technologien verschrieben (BPEL, WS-Security etc.). Für diese Arbeit wurde der Web Sphere Integration Server verwendet, um die Workflows zu modellieren und den erstellten Prozess ablaufen zu lassen.

Name des Produktes	Hersteller	Weitere Informationen unter
WebSphere Business Integration Server	IBM	http://www-306.ibm.com/software/info/ecatalog/de_DE/category/SW900.html?&S_TACT=none&S_CMP=none
BizTalk Server 2006	Microsoft	http://www.microsoft.com/biztalk/default.mspx
BizTalk Server 2006	Microsoft	http://www.microsoft.com/biztalk/evaluation/datasheet.mspx
BEA WebLogic Integration	BEA	http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/integrate
Oracle Fusion Middleware , Oracle BPEL Process Manager	Oracle	http://www.oracle.com/technology/products/integration/index.html

Abbildung 15: Übersicht evaluierter WfMS

3.3.1 BPEL

Die Business Process Execution Language for Web Services (BPEL4WS [ACD+03]), oder wie sie in diesem Dokument genannt wird BPEL, ist eine Sprache, die es ermöglicht, aus einer Komposition von einzelnen Web Services wiederum einen „Komposit“ Web Service zu erstellen. Dies ist aber nur eine der Möglichkeiten die BPEL zur Verfügung stellt. BPEL ermöglicht es, aus grundsätzlich zustandslosen Web Services eine zustandsbehaftete Komposition von Web Services zu erstellen. Dabei wird die zeitliche Abfolge der „aufgerufenen“ Web Services/Aktivitäten festgelegt sowie der Datenfluss zwischen den verschiedenen Prozesspartnern geregelt. Eine BPEL Komposition besteht einerseits aus einer BPEL Datei, die den Kontroll- und Datenfluss darstellt sowie einer WSDL (s. Abschnitt 3.2.2) Datei, die die PartnerLinkTypen sowie die benötigten Datenstrukturen beschreibt. Sie stellt die Schnittstellendefinition des Prozesses nach außen dar.

Wie bereits erwähnt, werden BPEL Kompositionen in einer XML Datei gespeichert, die die Endung .bpel hat. Da BPEL auf XML-Schema 1.0 [FWT+04] (Standardnamespace `http://www.w3.org/2001/XMLSchema`) basiert, muss ein Standard Namespace sowie ein Target Namespace vergeben werden. Der TargetNamespace repräsentiert den Namespace, durch den die BPEL Spezifikation von außen erreicht werden kann. Der „Standard Namespasce“, in dem BPEL spezifiziert ist, lautet
`http://schemas.xmlsoap.org/ws/2003/03/business-process/`. Hier sind alle in BPEL erlaubten Konstrukte als XML-Schema Datei spezifiziert. Um Bedingungen zu formulie-

ren sowie XML Dokumente zu durchsuchen, wird standardmäßig auf XPath 1.0 [W3C99] zurückgegriffen.

Bei der Betrachtung von BPEL muss man, um Missverständnissen vorzubeugen, zwischen zwei Begriffen mit verschiedenen Bedeutungen unterscheiden. Einerseits der Orchestrierung, andererseits der Choreographie. Beide Begriffe liegen dabei sehr nahe beieinander. Während eine Orchestrierung die Komposition einzelner Komponenten zu einem größeren Ganzen beschreibt, liegt die Betonung einer Choreoraphie eher auf der Definition der Nachrichten, die zwischen den einzelnen Partnern eines Prozesses ausgetauscht werden. BPEL ist eine „prozesszentrische“ Sprache. Das heißt, Dreh- und Angelpunkt ist der Prozess und nicht die Partner, die ihn realisieren. Daher beschreibt die BPEL Spezifikation einige Konstrukte, die wichtige Elemente einer Komposition darstellen. Jedes BPEL Dokument beginnt mit einem XML Element namens <process> (siehe nachfolgendes Codebeispiel). Darin müssen, wie oben beschrieben, der targetNamespace und der standardNamespace (xmlns=““) angegeben werden. Weiterhin sollte der Name des Prozesses im *name*-Attribut festgelegt werden. Falls eine von der oben genannten Standardsprache XPath 1.0 abweichende Anfragesprache verwendet werden soll, muss diese im Attribut queryLanguage spezifiziert werden. Auf das supressJoinFailure Attribut wird am Schluss dieses Abschnittes eingegangen. Manche Prozessmaschienen bieten die Möglichkeit, komplette Prozessinstanzen nach einer erfolgreichen Beendigung rückgängig zu machen. Dies wird durch das Attribut enableInstanceCompensation gesteuert. Um anzuzeigen, dass dies möglich ist, wird das Attribut auf „yes“ gesetzt. Default Wert ist „no“.

```
<process name="ncname" targetNamespace="uri"
         queryLanguage="anyURI"?
         expressionLanguage="anyURI"?
         suppressJoinFailure="yes|no"?
         enableInstanceCompensation="yes|no"?
         abstractProcess="yes|no"?
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <!-- Auf die folgenden Bestandteile wird später detailliert eingegangen -->
  <partnerLinks/>?

  <partners/>?

  <variables/>?

  <correlationSets/>?

  <faultHandlers/>?
  <compensationHandler/>?
  <eventHandlers/>?

  activity
</process>
```

Die in dem obigen Code Beispiel aufgeführten Konstrukte werden im Folgenden noch einmal kurz aufgelistet und dann jeweils ausführlicher beschrieben.

- `PartnerLinks` - BPEL bietet anhand dieses Konstruktes die Möglichkeit, Typen von Beziehungen zwischen Prozesspartnern zu definieren.
- `Variablen` - definieren die im Prozess verwendeten Datentypen und Nachrichten

- `Aktivitäten` - sind die elementaren Bausteine des Prozesses
- `CorrelationSets` - Korrelation - BPEL bietet anhand von correlationSets die Möglichkeit, einzelne Prozessinstanzen, anhand von Daten innerhalb der Nachrichten, die an den Prozess geschickt werden, eindeutig zu identifizieren.
- `Handler` - reagieren auf Ereignisse. So können in Prozessdefinitionen Nachrichtenhandler, Fehlerhandler und Timer verwendet werden, um auf festgelegte Ereignisse reagieren zu können.

PartnerLinks - Jede Interaktion zwischen dem Prozess und einem Partner wird durch einen PartnerLink festgelegt. In PartnerLinks wird die Rolle festgelegt, die der Prozess bzw. sein Partner einnimmt. Die Definition der verwendeten PortTypes sowie der verwendeten Nachrichten werden in dem zum Prozess gehörigen WSDL Dokument als PartnerLinkType gespeichert.

```
<partnerLinks>
   <partnerLink name="ncname" partnerLinkType="qname"
                myrole="ncname"? partnerRole="ncname"?>+
   </partnerLink>
</partnerLinks>
```

Variablen beschreiben die im Prozess verwendeten Datenstrukturen. Entweder in Form von WSDL Nachrichtentypen (messageTypes) oder als XML Schema. Eine explizite Definition der Datenstrukturen wird in der dem Prozess zugeordneten WSDL Datei vorgenommen.

```
<variables>
   <variable name="ncname" messageType="qname"?
             type="qname"? element="qname"? />+
</variables>
```

Auf ***Aktivitäten*** wird später im Detail eingegangen.

CorrelationSets dienen der Prozessinstanz Identifikation auf Basis der ausgetauschten Nachrichten. CorrelationSets bestehen aus einer Reihe von Eigenschaften (properties), die Nachrichtenteilen zugeordnet werden können. Um über verschiedene Nachrichten hinweg eine Identifikation zu ermöglichen, werden sogenannte property aliases definiert. Somit „zeigen“ correlationSets auf bestimmte „globale“ properties. Je nach Nachrichtentyp zeigen dann die den properties zugeordneten property aliases auf genau einen Teil einer bestimmten Nachricht. So können Informationen wie beispielsweise „Bestellnummer“ über verschiedene Nachrichtentypen (beispielsweise Bestellung/Rechnung) anhand nur eines properties identifiziert werden.

```
 <correlationSets>
     <correlationSet name="ncname" properties="qname-list" />+
 </correlationSets>
```

Handler - Es gibt drei verschiedene Arten von „Handlern“, die in einem BPEL Prozess vorkommen können.
So kann mit dem faultHandlers Konstrukt eine Fehlerbehandlung ausgeführt werden. Dabei kann für jeden Fehlerfall, der vorgesehen ist, eine eigene Fehlerbehandlung durchgeführt werden (catch). Mit dem catchAll Element können Fehler verarbeitet werden, die sonst in keiner catch Aktivität spezifiziert werden.

```
<faultHandlers>?
    <!-- Note: There must be at least one fault handler or default. -->
    <catch faultName="qname"? faultVariable="ncname"?>*
      activity
    </catch>
    <catchAll>?
      activity
    </catchAll>
</faultHandlers>
```

Mit ***eventHandlern*** kann auf in- oder externe Ereignisse reagiert werden. Anhand der onMessage Konstrukte können verschiedene Nachrichten spezifiziert werden, auf die der Prozess reagieren soll. Sobald eine der durch partnerLink, portType und operation spezifizierten Nachrichten eintreffen, werden die in dem onMessage Element deklarierten Aktivitäten ausgeführt. Anhand des onAlarm Konstruktes können zu einem bestimmten Zeitpunkt (until=“deadline-expr“) festgelegte Aktivitäten abgearbeitet werden. Es kann aber auch eine spezifizierte Zeitspanne(for=“duration-expr“) gewartet werden, bevor Aktivitäten ausgeführt werden.

```
<eventHandlers>?
    <!-- Note: There must be at least one onMessage or onAlarm handler. -->
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname" variable="ncname"?>
      <correlations>?
        <correlation set="ncname" initiate="yes|no"?>+
      <correlations>
      activity
    </onMessage>
    <onAlarm for="duration-expr"? until="deadline-expr"?>*
      activity
    </onAlarm>
</eventHandlers>
```

compensationHandler - Um bei einem Fehler die Resultate einer oder mehrerer Aktivitäten rückgängig zu machen, gibt es das compensationHandler Konstrukt. Wird es direkt nach einem scope oder flow angewendet, so bezieht sich der compensationHandler auf jeweils den Geltungsbereich, den der scope oder flow eingrenzt. Es ist jedoch auch möglich, für jede einzelne Aktion einen separaten compensationHandler anzugeben. Dazu wird er beispielsweise innerhalb eines invoke Konstruktes deklariert.

```
<compensationHandler>?
    activity
</compensationHandler>
```

Alle Handler sind über den kompletten Lebenszyklus eines Prozesses aktiv, wenn sie direkt nach dem process Element deklariert werden. EventHandler können sogar mehrmals parallel ausgeführt werden, falls mehrere spezifizierte Nachrichten kurz nacheinander ankommen.

Wie bereits vorher erwähnt, sind die Grundbausteine eines Prozesses **Aktivitäten**. BPEL unterscheidet hierbei zwischen **Basisaktivitäten** und **strukturierten Aktivitäten**.

3.3.1.1 Basisaktivitäten

Dieser Abschnitt beschriebt die in BPEL vorkommenden, verschiedenen Arten von „Basisaktivitäten“. Folgende Auflistung fasst diese zusammen. Die einzelnen Aktivitäten werden dann nocheinmal detaillierter beschrieben.

- **receive** - stellt eine Aktivität dar, die eine Nachricht von einem PartnerDienst empfängt.
- **reply** - stellt die Antwort auf eine von einem „Client“ empfangene Nachricht dar.
- **invoke** - dieses Konstrukt stellt die Ausführung von request/response oder one-way Operationen von WS dar, die von einem PartnerService zur Verfügung gestellt werden.
- **empty** - führt nichts aus.
- **assign** - bietet die Möglichkeit, Daten verschiedenen Prozessvariablen zuzuweisen.
- **wait** - lässt den Prozess eine definierte Zeit warten, bevor er mit seiner Ausführng fortfährt.
- **throw** - zeigt einen Fehler an, ähnlich dem throws Konstrukt aus Java.
- **terminate** - beendet den kompletten Prozess.

receive - Die Receive Aktivität stellt meist den Ausgangspunkt eines Prozesses dar. Sie wird verwendet, um das Aufrufen einer in der WSDL Datei des Prozesses spezifizierten Operation zu modellieren. Die empfangene Nachricht wird bei einem ausführbaren Prozess in der mit dem Feld variable spezifizierten Datenstruktur gespeichert. Das Feld createInstance wird verwendet, um zu indizieren, ob eine neue Prozessinstanz erstellt werden soll. Falls es auf „yes“ gesetzt wurde, muss ein correlationSet zur Identifikation der Prozessinstanz angegeben werden.

```
<recieive partnerLink="ncname" portType="qname" operation="ncname"
          variable="ncname"? createInstance="yes|no" standard-attributes>
    standard-elements
    <correlations>?
        <correlation set="ncname" initiate="yes|no"?/>+
    </correlations>
</recieive>
```

reply - Stellt die Antwortaktivität zur *receive* Aktivität dar. Diese Art der Modellierung entspricht einem synchronen Aufruf des Gesamtprozesses durch den Client. Der Prozess antwortet mit der in der Variable `variable` festgelegten Nachricht. Welche Operation zu diesem Zwecke benutzt werden soll, wird durch den spezifizierten partnerLink sowie den portType und operation festgelegt. Um Fehler innerhalb eines Prozesses an seine Prozesspartner zu übermitteln, können auch Fehlernachrichten („faultName“), die dann vom Partner verarbeitet werden müssen, versendet werden.

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
       variable="ncname" faultName="qname"? standard-attributes>
    standard-elements
    <correlations>?
        <correlation set="ncname" initiate="yes|no"?/>+
    </correlations>
</reply>
```

invoke - Ist die wohl meistbenutzte Aktivität in einer Prozessdefinition, die in BPEL erstellt wurde. Sie wird verwendet, um einen Web Service Partner aufzurufen. Dabei kann dieser in einer WSDL Datei spezifizierte Service asynchron oder synchron aufgerufen werden. Der Unterschied zwischen asynchronem und synchronem Aufruf manifestiert sich in der Verwendung der input- bzw. outputVariable. Wird nur die inputVariable definiert, wird an den Partner die spezifizierte Nachricht gesendet. Dies entspricht einem asynchronen Aufruf, da der Prozess nicht auf eine Antwort des Services wartet, sondern mit der Ausführung des Prozesses fortfährt. Dem entgegen wartet der Prozess, wenn zusätzlich eine outputVariable angegeben wird, bis eine Antwort vom Partner gesendet wurde. In diesem Falle muss auch ein correlation Pattern angegeben werden, das spezifiziert, ob die „Korrelation“ nur bei der ausgehenden (out), nur bei der eingehenden (in), oder bei beiden Nachrichen (out-in) angewandt werden soll.

```
 <invoke partnerLink="ncname" portType="qname" operation="ncname"
         inputVariable="ncname" outputVariable="ncname" standard-attributes>
    <correlations>?
        <correlation set="ncname" initiate="yes|no"? pattern="out|in|out-in"/>+
    </correlations>
    <catch faultName="qname" faultVariable="ncname"?>*
       activity
    </catch>
    <catchAll>?
       activity
    </catch>
    <compensationHandler>?
       activity
    </compensationHandler>
</invoke>
```

empty - Die empty-Aktivität führt keine Aktion aus. Sie kann beispielsweise dazu benützt werden, um in einer Fehlerbehandlung nichts auszuführen oder bestimmtes Kontrollflußverhalten nachzubilden.

```
<empty standard-attributes>
   standard-elements
</empty>
```

assign - Diese Aktivität dient dazu, Prozessvariablen zu initialisieren oder Werte zwischen verschiedenen Variablen zu kopieren. Mit ihrer Hilfe können auch Endpunktreferenzen kopiert werden.

```
<assign standard-attributes>
   standard-elements
   <copy>+
     from-spec
     to-spec
   </copy>
 </assign>
```

wait - Dieses Konstrukt dient dazu, eine bestimmte Zeitspanne zu warten (for="duration-expression") oder die Ausführung des Prozesses bis zum Eintreten eines Zeitpunktes (until="deadline-expr") auszusetzen.

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
    standard-elements
</wait>
```

throw - Wirft explizit einen in der faultVariable spezifizierten Fehler, der von dem Faulthandler des umschließenden Scopes (sequence, scope, flow) behandelt werden sollte. Ist eine Fehlerbehandlung nicht in dem nächsten umschließenden Element möglich, so wird der Fehler so lange weiter nach „oben" gereicht, bis er den globalen Fault Handler erreicht. Wird er in diesem nicht behandelt, terminiert der ganze Prozess, samt seiner Subprozesse.

```
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
    standard-elements
</throw>
```

terminate - Diese Aktivität beendet sofort die komplette Prozessinstanz mit allen ihren Subprozessen. Dabei wird kein Fehler erzeugt. Man kann die terminate Aktivität auch als explizites reguläres Ende eines Prozesses betrachten.

```
<terminate standard-attributes>
    standard-elements
</terminate>
```

3.3.1.2 Strukturierte Aktivitäten

Um in der Logik der Prozessausführung die nötige Flexibilität zu erreichen, die von einer Sprache erwartet wird, die für das „programming in the large" steht, müssen auch einige den Kontrollfluss beeinflussende Konstrukte zur Verfügung stehen. Diese werden Strukturierte Aktivitäten genannt.

- `sequence` - In einer sequence werden einzelne Aktivitäten nacheinander in der Reihenfolge ausgeführt, in der sie in der Sequence spezifiziert sind.

- `switch` - Wie aus vielen Programmiersprachen bekannt, steht das switch Konstrukt für die konditionelle Ausführung von Aktivitäten, die mit einem boolschen Vergleich versehen sind. Dabei kann für jeden Fall eine zugehörige Aktivität spezifiziert werden. Weiterhin kann auch eine Standardaktivität festgelegt werden, welche ausgeführt wird, falls keine der boolschen Ausdrücke mit Ihren Aktivitäten zu „wahr“ evaluiert wird.
- `pick` - Die pick Aktivität reagiert auf Ereignisse. Ein solches Ereignis kann beispielsweise eine Nachricht sein, die an den Prozess gesendet wird oder das Ablaufen eines Timers. Jedem Ereignis wird eine Aktivität zugeordnet und nachdem das korrespondierende Ereignis eingetreten ist, ausgeführt.
- `while` - Die Aktivität die mit diesem Konstrukt versehen ist, wird so lange ausgeführt, bis der dazugehörige boolsche Ausdruck als „unwahr“ evaluiert wird. Dabei kann die Aktivität, die mit dem Konstrukt versehen ist, wiederum eine strukturierte sein (beispielsweise eine Sequenz).
- `flow` - Ähnlich wie die sequence strukturiert der flow einzelne Aktivitäten. Im Unterschied zur sequence ist hier jedoch eine parallele Ausführung möglich. Daher müssen jedoch in diesem Fall die Kontrollflußkanten explizit angegeben werden. So ist die sequence (ABC), also C nach B nach A, gleichbedeutend mit dem flow (A,B,C(Kanten(AB,BC))).

sequence - Stellt eine Sequenz von Aktivitäten dar, die nacheinander ausgeführt werden. Die Reihenfolge der Ausführung der einzelnen Aktivitäten wird durch ihr Vorkommen in dem sequence Konstrukt festgelegt. Dabei wird das „oberste“ Element als erstes ausgeführt und das „unterste“ als letztes.

```
<sequence standard-attributes>
    standard-elements
    activity+
</sequence>
```

switch - Das aus der Programmierung bekannte switch Konstrukt erlaubt es, aus einer Menge von Alternativen, die durch ein <case> Konstrukt gekennzeichnet sind, genau eine zutreffende Alternative zu wählen. Jedes <case> Konstrukt ist dabei mit einer Bedingung ausgestattet. Wird die Bedingung zu true (wahr) ausgewertet, werden die spezifizierten Aktivitäten ausgeführt. Innerhalb eines <case> Konstruktes können wiederum strukturierte oder unstrukturierte Aktivitäten definiert werden. Falls keine der durch <case> spezifizierten Aktivitäten zutrifft, kann durch die Angabe des <otherwise> Konstruktes eine default Aktivität angegeben werden.

```
<switch standard-attributes>
  standard-elements
  <case condition="bool-expr">+
    activity
  </case>
  <otherwise>?
    activity
  </otherwise>
</switch>
```

pick - Die pick Aktivität dient dazu, auf eine aus einer Auswahl von externen Nachrichten oder das Verstreichen von deadlines zu warten. Im Gegensatz zu „event handlern“, die über den kompletten Lebenszyklus einer Prozessinstanz hinweg auch konkurrent auf externe Ereignisse reagieren, hat die pick Aktivität einen sehr viel kürzeren Lebenszyklus. Durch das Setzen des createInstance Flags auf yes, kann pick am Anfang einer Prozessdefinition auf das Eintreffen von Nachrichten warten und eine Prozessinstanz erzeugen. Ist das createInstance flag gesetzt, darf jedoch kein <onAlarm> event spezifiziert werden. Weiterhin muss in diesem Fall wie bei einer instanzauslösenden <receive> Aktivität ein correlation set angegeben werden, um eine Prozessinstanz Identifikation zu ermöglichen. Bei dem <onAlarm> Element kann entweder eine Zeitdauer mithilfe des „duration-expr“ Konstruktes spezifiziert, oder auf das Eintreten eines Zeitpunktes („deadline-expr“) reagiert werden.

```
<pick createInstance="yes|no"? standard-attributes>
    standard-elements
    <onMessage partnerLink="ncname" portType="qname"
               operation="ncname" variable="ncname"?>+
      <correlations>?
         <correlation set="ncname" initiate="yes|no"?>+
      </correlations>
      activity
    </onMessage>
    <onAlarm (for="duration-expr" | until="deadline-expr")>*
      activity
    </onAlarm>
</pick>
```

while - Die <while> Aktivität ist die einzige Aktivität in einem BPEL Prozess, anhand derer die wiederholte Ausführung einzelner Aktivitäten spezifiziert werden kann. Die in ihr geschachtelten Aktivitäten werden so oft wiederholt, bis die Schleifenbedingung (condition) zu false (falsch) evaluiert wird. Diese Aktivität hat genau einen Ein- und Ausgang. Es ist in der jetzigen Version von BPEL nicht möglich, wie aus der Programmierung den geplanten Abbruch des Schleifendurchlaufes anhand eines Konstruktes (beispielsweise „break“) zu modellieren.

```
<while condition="bool-expr" standard-attributes>
     standard-elements
     activity
</while>
```

flow - das Flow Konstrukt ist wohl eines der mächtigsten Konstrukte einer BPEL Komposition. Es wurde eingeführt, um Nebenläufigkeit/Parallelität modellieren zu können. Mit der flow Aktivität lassen sich gerichtete azyklische Graphen (directed acyclic graphs DAGs) konstruieren. Aktivitäten stellen hierbei die Knoten dar, während die so genannten „*links*“ die Kanten zwischen den Knoten repräsentieren. Eine Kante darf jedoch nie auf einen Vorgänger ihres Ausgangsknotens zeigen, da hierdurch Zyklen entstehen würden, die nicht erlaubt sind. Die link Konstrukte selbst besitzen keine Informationen, welcher Knoten/Aktivität ihr Ziel bzw. ihren Ausgangspunkt darstellen. Dies wird in den Aktivitäten anhand der Konstrukte *target* (Ziel) und *source* (Ausgangpunkt) festgelegt. Links können jedoch eine sogenannte *transitionCondition* beinhalten, die eine Bedingung

darstellt, wann oder ob diese Kante bei der Ausführung „traversiert“ werden kann. Die transitionCondition wird ausgewertet, sobald die Aktivität, die den Ausgangspunkt des links darstellt, „abgearbeitet“ ist. Jeder Link darf nur von genau einer Aktivität als Ziel, beziehungsweise als Ausgangspunkt definiert werden. Um innerhalb eines flow Konstruktes die Synchronisation von Aktivitäten realisieren zu können, bieten Aktivitäten, die Ziel mindestens eines links sind, die Möglichkeit, eine sogenannte *joinCondition* zu spezifizieren. Durch eine joinCondition kann festgelegt werden, ob beispielsweise alle eingehenden Ausführungsäste erfolgreich durchlaufen werden müssen, oder ob es reicht, dass nur einer der eingehenden Äste erfolgreich durchlaufen wird. Zu diesem Zwecke haben links einen impliziten Status.

- positiv - (true) die Ausgangsaktivität des Links ist abgeschlossen und die transitionCondition ist erfüllt
- negativ - (false) die Ausgangsaktivität des Links ist abgeschlossen, aber die transitionCondition wurde nicht erfüllt.
- undefined - die Ausgangsaktivität ist noch nicht ausgeführt, daher kann keine Aussage über den Status des Links gemacht werden.

Um bei einer joinCondition auf den Status eines Links zuzugreifen, bietet BPEL eine Methode getLinkStatus(„linkName“). Diese Funktion gibt true zurück, falls der Status des links positiv ist und false, falls er negativ ist. Sie wird im Standard Namespace von BPEL (`http://schemas.xmlsoap.org/ws/2003/03/business-process`) definiert. **Abbildung 16** veranschaulicht die Zusammenhänge der genannten Funktionalitäten noch einmal grafisch.

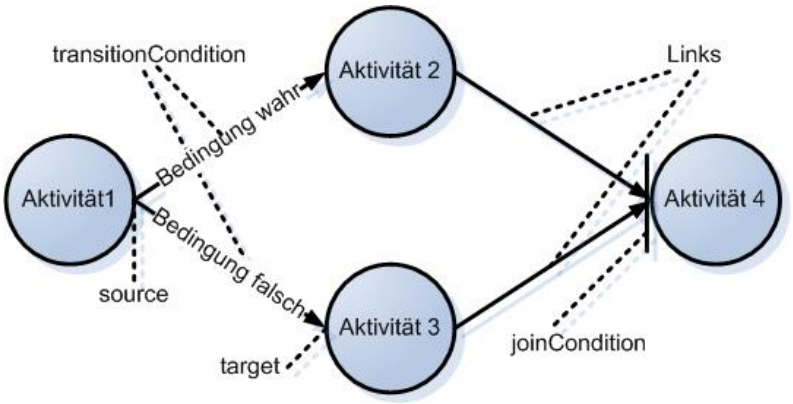


Abbildung 16: Veranschaulichung des flow Konstruktes

Die Standard Belegung der joinCondition, auch wenn diese nicht explizit definiert ist, ist die „Veroderung“ der Stati aller eingehenden Links. Da der Status eines Links erst geprüft werden kann, wenn die Ausgangsaktivität des Links abgeschlossen ist, wurde es nötig, die so genannte dead-path-elimination (DPE) einzuführen, auf die nach dem folgenden Code-Exzerpt eingegangen wird. Die Start Aktivitäten eines flow Konstruktes charakterisieren sich dadurch, dass sie keine eingehenden Kanten besitzen.

```
<flow standard-attributes>
    standard-elements
    <links>?
      <link name="ncname">+
    </links>
    activity+
</flow>
```

JoinFailures und die dead-path-elimination - Wie oben bereits erwähnt, können bei Aktivitäten, die das Ziel mindestens eines links sind, so genannte joinConditions spezifiziert werden. Diese Bedingungen werden vor der Ausführung der Aktivität, in der sie spezifiziert sind, ausgewertet. Wird diese Bedingung nicht erfüllt, wird standardmäßig ein Fehler (joinFailure) verursacht. Selbst ohne eine joinCondition zu spezifizieren, könnte ein Fehler geworfen werden, falls der Status aller eingehenden links negativ ist. Dies rührt daher, dass die Standardbelegung der joinCondition die Veroderung aller eingehenden links ist. Um ein solches Verhalten zu unterbinden, kann anhand des Attributes supressJoinFailure, das auf „yes“ gesetzt wird, diese Funktionsweise verhindert werden. Die default Belegung dieses Attributes ist „no“ (nein). Die Konsequenz, die sich daraus ergibt, wenn suppressJoinFailure auf „yes“ gestellt wird, und die joinCondition zu falsch evaluiert wird, ist, dass die Aktivität nicht ausgeführt wird und die aus ihr führenden Links den Status „negativ“ erhalten. Dies wird entlang des Kontrollflusses so lange fortgeführt, bis mindestens ein an einer Aktivität ankommender link, den Status „undefined“ hat oder eine joinCondition zu „true“ evaluiert wird. Dieses Verhalten wird dead-path-elimination genannt [LR99].

In den oben gezeigten Code-Auszügen wird der Terminus ***standard-attributes*** ersetzt durch folgenden Codeabschnitt:

```
name="ncname"?
 joinCondition="bool-expr"?
 suppressJoinFailure="yes|no"?
```

standard-elements steht für:

```
<target linkName="ncname"/>*
<source linkName="ncname" transitionCondition="bool-expr"?/>*
```

3.4 Vorgehensmodelle für Softwareentwicklung

In den folgenden Abschnitten werden kurz drei Vorgehensmodelle für die Softwareentwicklung vorgestellt. In Kapitel 4 werden dann die Vor- und Nachteile erörtert und eines dieser Modelle für das Vorgehen in dieser Arbeit herangezogen.

3.4.1 Das Spiralmodell

Das Spiral Modell, das Barry W. Boehm in seinem Artikel „A Spiral Model of Software Development and Enhancement“ [Boe88] vorstellte, fußt auf einem standardisierten Iterativen Prozess. Dabei wird das Gesamtprojekt in verschiedene Phasen aufgeteilt, die jeweils wiederum in eine feste Abfolge von Schritten untergliedert werden. Die Schritte, die jede Phase dieses Iterativen Vorgehensmodells durchlaufen müssen, sind folgende:

1. Festlegung von Zielen, Alternativen und Rahmenbedingungen
2. Evaluation der Alternativen und das Erkennen und Reduzieren von Risiken
3. Realisierung und Überprüfung des Zwischenproduktes
4. Planung der Projektfortsetzung

Diese Schritte oder Phasen werden für jeden iterativen Schritt bis zur Fertigstellung des Produktes durchlaufen.

3.4.2 Der Rational Unified Process

Der Rational Unified Process [KK03] wurde, als Verfeinerung des Unified Process, von der Rational Software Corporation entworfen. Beide Verfahren basieren auf dem im vorigen Abschnitt beschriebenen Spiralmodell von Boehm. Somit sind auch sie Vertreter der iterativen Entwicklungsprozesse. Dabei fließen die Erfahrungen aus den vorherigen Iterationen in die nächste Iteration mit ein. Der RUP teilt dabei jedes Softwareentwicklungsprokjekt in 4 Phasen ein.

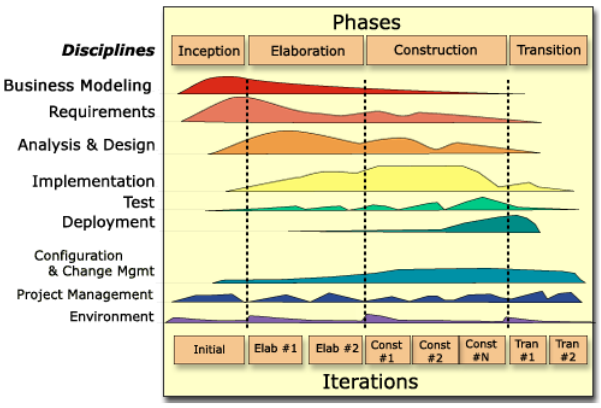


Abbildung 17: Der Rational Unified Process [KK03]

1. Inception (Einstiegs Phase)
2. Elaboration (Ausarbeitungs Phase)
3. Construction (Konstruktions Phase)
4. Transition (Überleitungs Phase)

3.4.3 Das Phasenmodell nach Molters

Das von Michael Molter in seinem Artikel „Die prozessorientierte Applikationslandschaft“ [SJW05] vorgestellte, spezialisierte „Software Lifecycle“ Modell versucht, einen ganzheitlichen Integrationsansatz zu verwirklichen. Anders wie die beiden vorher vorgestellten Entwicklungsverfahren, ist dieses Software Lifecycle Modell auf die Integration und Konsolidierung einer Anwendungssystemlandschaft ausgerichtet. Dabei werden in der ersten Phase neue Geschäftsideen oder vorhandene Geschäftsprozesse definiert und modelliert. Dabei werden diejenigen Prozesse, die sich für eine IT-technische Umsetzung eignen, durch Anforderungsspezifikation präzisiert. Danach wird eine Modellierung der beteiligten IT-Landschaft sowie der benutzten Datenstrukturen umgesetzt. Aus den Informationen der Modelle der vorherigen Phasen wird dann eine Software erstellt, die dann gegen die Anforderungen des Geschäftsprozesses getestet wird und in einem nächsten Schritt in die bestehende Anwendungslandschaft integriert wird. Nach diesen Schritten steht einem Betrieb nichts mehr im Wege. Dabei ist darauf zu achten, dass durch Monitoring der Applikation sowie wenn möglich Benchmarking, Informationen gesammelt werden, die ein eventuelles Optimierungspotenzial offenbaren. Falls dies der Fall ist, wird der Kreis von neuem durchlaufen. **Abbildung 18** zeigt diesen Zyklus.

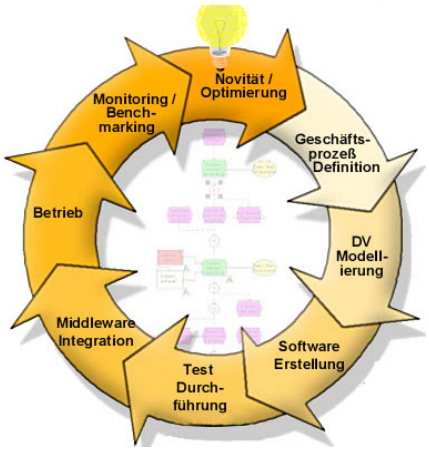


Abbildung 18: Lifecycle Modell [SJW05]

4 Vorgehensweise

Um den in Kapitel 2.1.2 beschriebenen Ansprüchen der Unternehmen an eine IT Infrastruktur gerecht zu werden, müssen die eingesetzten IT Systeme in demselben hohen Maße flexibel sein wie die Unternehmen selbst. Erfolgsfaktoren für Unternehmen, die prozessorientiert aufgebaut sind, sind demnach:

- Minimierung der Ausführungszeit und Kosten der Prozesse
- Maximierung der Anzahl der ausgeführten Prozesse

Ein Werkzeug, das sich im betriebswirtschaftlichen Umfeld etabliert hat, um Prozesse sowie Organisationsstrukturen zu beschreiben, sind die in Kapitel 3.1.1 beschriebenen ereignisgesteuerten Prozessketten. Viele Geschäftsprozesse, die im Zuge sogenannter BPE/BPR Projekte durch die vorhandene IT Struktur unterstützt werden sollen, liegen in Unternehmen schon als EPK Modelle vor. Daher wird in dieser Arbeit auf diese Form der Geschäftsprozessmodellierung als Ausgangsform für Integrationsprojekte zurückgegriffen.

Die „ambivalente Rolle“ [SJW05], einerseits die Prozesseffizienz zu erhöhen, andererseits durch Komplexität den Wandel der unterstützten Geschäftsprozesse zu erschweren, die IT als wichtiger Bestandteil der Unternehmensstrategie heutzutage spielt, muss überwunden werden.

Eine Möglichkeit, diese Hürde zu überwinden, bietet die Service-orientierte Architektur. Sie verringert die Komplexität der Anwendungslandschaft und bietet eine hohe Flexibilität. Wie in Kapitel 3.2 beschrieben, bedient sich die Service-orientierte Architektur dafür der Techniken der losen Kopplung sowie der festen Schnittstellen bzw. der vertragsbasierten Entwicklung. Dies resultiert in einer hochflexiblen komponentenorientierten Architektur, deren einzelne Bestandteile leicht in neue Prozesse integriert werden oder durch neue Module ersetzt werden können.Genau diese oben genannte und geforderte Flexibilität und Anpassbarkeit sind die Stärken einer Service-orientierten Architektur. In dieser Arbeit wird daher der Ansatz der Service-orientierten Architektur eingesetzt, der im Moment auf die breiteste Unterstützung in Wissenschaft und Industrie stößt.Um eine Interoperabilität über die Grenzen eines Unternehmens hinaus gewährleisten zu können, ist eine andere Anforderung, die gestellt werden muss, die Nutzung verbreiteter Standards. Es gibt viele Möglichkeiten eine Service-orientierte Architektur zu realisieren. Die im Moment am weitesten verbreitete ist die der Web Service Architektur. Die meisten für geschäftliche Interaktionen wichtigen Kriterien sind bereits standardisiert, oder sind als Drafts den Standardisierungsgremien wie OASIS oder W3C übergeben worden. Da die Standards von den größten Herstellern wie SAP, IBM, Microsoft, Siebel und Oracle vorangetrieben werden, kann davon ausgegangen werden, dass eine Interoperabilität über Herstellergrenzen hinweg gegeben sein wird. Daher wird in dieser Arbeit ein Web Service Design für die Realisierung gewählt.

Workflows, also technische Modelle von Geschäftsprozessen, sind mehr als nur die Nacheinanderausführung einzelner Komponenten. Es gibt eine Vielzahl von technischen Prozessbeschreibungssprachen. Jede dieser Sprachen wird meist jedoch nur von dem Produkt

und Hersteller unterstützt, für den sie entwickelt wurde. Wenn eine Service-orientierte Architektur mit Hilfe von WS Technologien umgesetzt wird, ist die Sprache der Wahl momentan BPEL. Dies rührt daher, dass auch dieser Standard durch eine breite Auswahl von Herstellern, die bei der Entwicklung des Standards mitgewirkt haben, unterstützt wird. Zu diesen Herstellern gehören die Branchenführer im EAI Markt wie BEA, SAP, IBM, Microsoft, Siebel und Oracle. Die meisten Hersteller bieten bereits heute einen Im- und Export von BPEL in ihre proprietären Produkte an.

4.1 Anforderungen an das Vorgehensmodell

Da bei dem Prozess der Anwendungsintegration sehr viele Nutzergruppen auf den verschiedenen Ebenen der Unternehmen beteiligt sind, werden in der folgenden Aufzählung die am Prozess der Anwendungsintegration/Prozessmodellierung beteiligten „Stakeholder“ aufgeführt.

- Management - Das Management erstellt die fachlich-inhaltlichen Prozessmodelle, vorgegeben durch den Markt und die daraus resultierenden strategischen Entscheidungen
- Systemarchitekten - Erstellen die für die Integration wichtigen Kopmpositionsschemata
- Softwareentwickler - Sie erstellen die notwendigen Komponenten oder ändern diese, so dass sie den Anforderungen des Geschäftsprozesses genügen
- Benutzer - Sie sind in den regulären Prozessablauf involviert und benutzen das entstandene System um eine Wertschöpfung für die Unternehmung zu erreichen.
- Externe Stakeholder - Sie sind nicht direkt in den Prozess der Erstellung oder der Nutzung des Systems eingebunden. Beispiele könnten von Anteilseignern über Prüfungsgesellschaften oder Auftraggeber reichen.

All diese Nutzergruppen müssen durch die in diesem Vorgehensmodell erstellten Modelle mit den für sie relevanten Informationen versorgt werden. Die durch die Modelle zur Verfügung gestellten Informationen sollten so aktuell wie möglich gehalten werden, um Missverständnisse in der Kommunikation zwischen den Partnern auszuschließen.

Um ein standardisiertes Vorgehen anzubieten, das von der fachlichen Geschäftsprozessdefinition bis zur fertigen Integrationslösung reicht, kommen einige etablierte Vorgehensmodelle in Frage. Diese wurden in Kapitel 3.4 beschrieben.

Alle drei in Kapitel 3.4 aufgeführten Prozesse haben einen wichtigen Punkt gemeinsam: Sie sind alle iterative Prozesse. Dies ist in der Geschäftsprozessmodellierung und ihrer Umsetzung in ausführbare Workflows der praktikabelste Ansatz, da ein Prozess in seiner erstmaligen Definition sich oftmals noch ändert oder noch Optimierungspotenzial bietet.

Der Rational Unified Process baut auf dem Spiral Modell von Boehm auf und ist für die Zwecke der Integration von Anwendungssystemen in kleinen oder mittelständischen Unternehmen, die eine Softwareentwicklungsabteilung unterhalten, die schon mit dem RUP arbeiten, etwas abgewandelt geeignet. Die Einführung und Umstellung auf eine Entwicklung nach dem RUP ist eine zeitintensive und langwierige Angelegenheit. Die Komplexität des Prozesses sollte nicht unterschätzt werden. Daher wird in dieser Arbeit ein einfacherer und schneller zu realisierender Ansatz gewählt.

Das Spiralmodell genügt den Anforderungen an eine Vorgehensweise im Bereich der prozessgetriebenen Anwendungsintegration nicht, da es im Jahre 1989 mit dem Ziel als Leitfaden zur Softwareentwicklung zu dienen, entwickelt wurde. Es könnte verfeinert werden, um den Anforderungen an Integrationsprojekte zu genügen. In der originalen Fassung werden jedoch die betriebswirtschaftlichen Aspekte und Vorraussetzungen, die eine IT Infrastruktur legitimieren und finanzieren, nicht berücksichtigt. Der Fokus liegt allein bei der Entwicklung von Anwendungssystemen. Dies ist jedoch nur ein Teilaspekt, der bei einer Integration von verteilten Anwendungssystemen zu betrachten ist. Der Aufwand, um das Spiralmodell zu einem praktikabelen Ansatz für die Umsetzung von Integrationslösungen umzugestalten, wäre, verglichen mit den Änderungen, die am vorgeschlagenen Vorgehen vorgenommen werden müssen, zu hoch.

Daher wird in dem Kapitel 4.2 ein Vorgehen geschildert, das sich an dem in Kapitel 3.4.3 vorgestellten spezialisierten „Software Lifecycle“ Modell orientiert, und dies um einige Aspekete erweitert.Der „round-trip“ Aspekt wird im Rahmen dieser Arbeit nicht weiter behandelt, da hierzu noch wesentliche Forschungsergebnisse fehlen, insbesondere im Hinblick auf die Synchronisation der unterschiedlichen Modelle. Problematiken, die im Rahmen dieser Arbeit identifiziert worden sind, aber nicht den Schwerpunkt dieser Arbeit darstellen, werden im Folgenden kurz angesprochen. In der vorgestellten Methodik, soll zwar eine Transformation von einem fachlichen, zu einem technischen Prozessmodell ermöglicht werden, die Retransformation zurück zum fachlichen Modell birgt jedoch einige Probleme. Es kann aus einem fachlichen Modell nur ein Gerüst des technischen Prozesses erstellt werden, das dann durch Spezialisten um technische Details erweitert werden muss. Wenn jedoch Optimierungspotenzial durch das Monitoring entdeckt werden sollte und das bestehende fachliche Modell optimiert wird, würden die manuell eingeführten Änderungen am technischen Modell bei einer erneuten Transformation des fachlichen in das technische Modell verloren gehen. Das Grundproblem sind die verschiedenen Abstraktionsebenen auf denen das fachliche beziehungsweise das technische Modell angesiedelt sind. Weiterhin verursachen die verschiedenen Werkzeuge, die zur Modellierung eingesetzt werden, weitere Probleme bei der Bestrebung ein „round-trip“ Verfahren anbieten zu können.

4.2 Das Phasenmodell der Prozessintegration

Das in dieser Arbeit verwendete spezialisierte Phasenmodell (s. **Abbildung 19**) lehnt sich an das von Michael Molter in [SJW05] veröffentlichte Modell an. Ein gravierender Unterschied ist eine zusätzliche Integrationstest- Phase nach der Integration der Middleware Schichten. Dies ist dadurch bedingt, dass aus getesteten und „validierten“ Teilkomponenten nicht zwingend folgt, dass ein funktionierendes ganzheitliches Anwendungssystem entsteht. Ein weiterer Unterschied zum Modell von Molters besteht darin, dass die Entwicklung und das Testing neuer Komponenten nur in Einzelfällen zum Einsatz kommt, da idealerweise bei einem Integrationsprojekt nur vorhandene Software in neuer Form miteinander genutzt wird. In einigen Fällen ist natürlich eine Erweiterung der bestehenden Softwarelandschaft unabdingbar. Diese „Spezialfälle“ werden jedoch in den einzelnen Unterkapiteln erörtert.

Der Schwerpunkt der Arbeit liegt dabei auf den Phasen der „Geschäftsprozess Analyse/Definition“, der „IT-Modellierung“ sowie der „Middleware/Komponenten Integration“. In **Abbildung 19** sind diese Phasen dementsprechend hervorgehoben. Die als Optional beziehungsweise Nebenaspekt gekennzeichneten Phasen werden nur rudimentär behandelt.

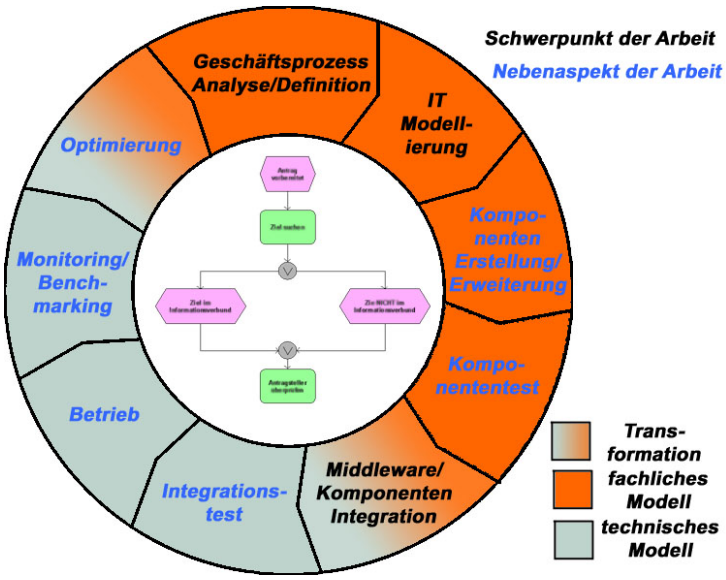


Abbildung 19: Abgewandeltes Phasenmodell Prozessorientierter Integration nach [SJW05]

Der Standardablauf einer Integration nach diesem Vorgehensmodell würde demnach, wie in **Abbildung 19** zu sehen, mit der Geschäftsprozessanalyse / Definition beginnen. Danach schließt sich eine Phase der Modellierung der beteiligten IT Systeme an. Falls in den ersten beiden Phasen ein Bedarf für zusätzliche Funktionalität festgestellt wurde, schließen sich die Phasen der Softwareentwicklung und des Komponententests an. Andernfalls werden diese beiden Phasen (In **Abbildung 19** als Nebenaspekt gekennzeichnet) über-

sprungen. Anschließend werden die vorhandenen und im Rahmen des Geschäftsprozesses notwendigen Middleware Plattformen integriert und der definierte Geschäftsprozess in einen technisch ausführbaren Prozess (beispielsweise BPEL) überführt. Nach dem darauf folgenden Integrationstest wird das neue System in Betrieb genommen. Der nächste Schritt sollte dann als fortlaufende Maßnahme gesehen werden. Das Monitoring des technischen Prozesses sowie das Benchmarking, soweit dies möglich ist, sind Aktivitäten, die sich über den kompletten Lebenszyklus eines Prozesses erstrecken sollten. Wenn durch das Monitoring und Benchmarking Änderungsbedarf erkannt wird, wird der Geschäftsprozess im nächsten Schritt optimiert und dann neu definiert. Eine ausführliche Beschreibung der einzelnen Phasen folgt.

4.2.1 Geschäftsprozess-analyse / definition

Die Geschätsprozess-analyse und definition bildet den zentralen Ausgangspunkt für das in diesem Kapitel beschriebene Vorgehen. Die Gründe für eine Analyse bestehender Geschäftsprozesse oder die Definition neuer Geschäftsprozesse sind sehr vielseitig. Die Definition von Geschäftsprozessen kann beispielsweise im Zuge der Steigerung der Unternehmenstransparenz betrieben werden. Ein weiterer wichtiger Grund für die Definition von Geschäftsprozessen kann eine Neuerschließung von Geschäftsfeldern oder Märkten sowie eine Kollaboration mit neuen Partnern sein.

Im Projekt Integration Engineering wurde eine Methode entwickelt und vorgestellt[DS05] [FTH05], die eine solche Modellierung unterstützt. Das entwickelte Vorgehen sollte als Grundlage dieser Modellierung herangezogen werden.Viele SMEs haben im Zuge von BPO bzw. BPR Projekten eine Vielzahl der vorhandenen Geschäftsprozesse schon anhand von ereignisgesteuerten Prozessketten modelliert. Bei der „Integration Engineering“ Methodik kann direkt auf diese bestehenden Modelle aufgesetzt werden. Da die ARIS-IE Methodik die Basis dieser Vorgehensweise darstellt, wird in diesem Vorgehensmodell das zugehörige ARIS Toolset dazu verwendet, den Geschäftsprozess zu erfassen. Das aus diesem Prozessschritt enstehende Modell sollte den Ablauf des Geschäftsprozesses aus einer fachlich-inhaltlichen Sichtweise dokumentieren und auf technische Details verzichtet um den Blick auf den eigentlichen Geschäftsprozess nicht zu verstellen. Diese eher technische Sichtweise auf den Geschäftsprozess wird in der nächsten Phase dieses Vorgehens (s. Abschnitt 4.2.2) ergänzt. Geschäftsprozessdefinitionen sollten aber zu mehr als lediglich dokumentarischen Zwecken dienen. Sie stellen das Leitbild für die weiteren Phasen und Schritte dar und bilden den Rahmen für die sinnvolle Verteilung von Informationen an die am Prozess beteiligten Akteure. Die Arbeit der Prozessdefinition sollte durch ein Team von spezialisierten Analysten sowie den direkt am Prozess beteiligten „stakeholdern“ ausgeführt werden. Inhalte des Modells sind:

- Die Aktivitäten und ihr zeitlicher Zusammenhang/Abfolge
- Die Organisationseinheiten / Rollen die den Aktivitäten zugeordnet sind

Dabei sollte unterschieden werden, ob eine Funktion manuell oder maschinengestützt abläuft.

Als typisches Beispiel für ein aus diesem Schritt entstandenes EPK Modell wird auf **Abbildung 20** verwiesen. Dabei ist zu beachten, dass alle Funktionen zu diesem Zeitpunkt noch nicht mit einem eFZD hinterlegt sein sollten, da der Fokus der Modellierung ganz klar auf der fachlich-inhaltlichen Sicht liegen sollte.

Die zu berücksichtigenden Einschränkungen, die sich an eine solche Modellierung der Geschäftsprozesse stellen, um den Geschäftsprozess in ein technisches Modell transformieren zu können sowie eine ausführliche Beschreibung werden in Kapitel 4.4 dargestellt.

Das entstandene Prozessmodell dient nun als Grundlage für die weiteren Prozessschritte, die zum Ziel haben, ein neues prozessunterstützendes Anwendungssystem zu integrieren. Zu diesem Zweck sind jedoch noch weitergehende Informationen vonnöten. Diese werden im nächsten Schritt, der IT-Modellierung (s. Abschnitt 4.2.2) des Prozessmodells erfasst.

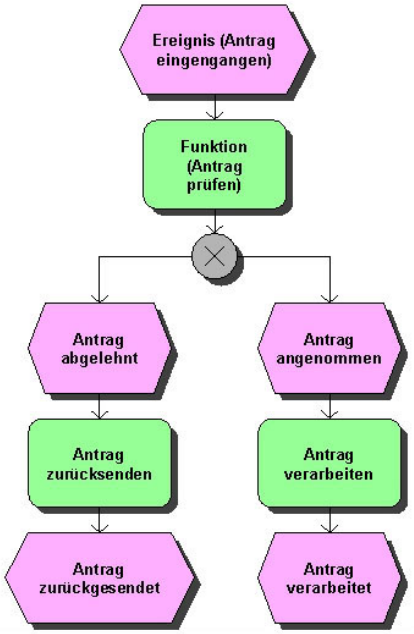


Abbildung 20: Beispiel einer EPK

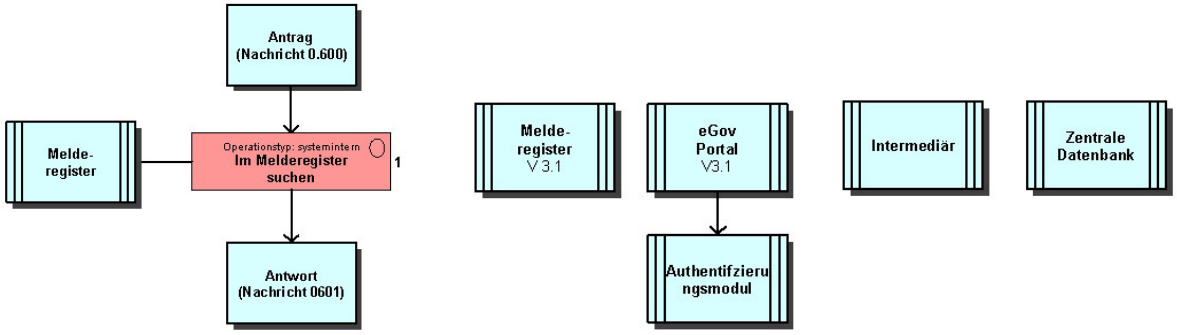
4.2.2 IT-Modellierung

Dieser Teil des Prozesses erweitert die fachlich-inhaltliche Geschäftsprozessdefinition um technische Details. Dazu bieten sich eine Vielzahl von Werkzeugen und Notationen an. So wäre eine Notation in UML 2.0 (s. Kpitel 3.1.4) eine sinnvolle Alternative zu der hier beschriebenen Form. Um jedoch die Anzahl der verwendeten Werkzeuge und die dadurch entstehende Problematik der redundanten Modellierung zu vermeiden, wird darauf verzichtet. Da im Rahmen des vorherigen Prozessschrittes schon das ARIS Toolset mit dem IE Methodenfilter zum Einsatz gekommen ist, wird dieses Werkzeug, in der in dieser Arbeit vorgeschlagenen Vorgehensweise, auch für diese Phase verwendet.

Um eine sinnvolle Unterstützung zukünftiger Projekte und Prozessdefinitionen zu gewährleisten, ist es unabdingbar, die am Geschäftsprozess beteiligten IT Systeme zu modellieren. Die in dieser Phase gesammelten Informationen stellen den wohl wichtigsten Teil für eine technische Transformation dar. In der hier vorgestellten Vorgehensweise müssen zusätzlich zu den Anwendungssystemen auch deren benutzte Operationen modelliert werden. Auch die verwendeten Entitätstypen (vergleichbar mit Nachrichten), die die Anwendungssysteme/Operationen untereinander austauschen, sollten, soweit sie bekannt sind, in einem Repository angelegt werden.

Einzelne Funktionen werden anhand der „Integration Engineering“ Methodik um Anwendungssysteme, Entitäten und Operationen sowie ihrer zeitlichen Abfolge ergänzt. Dies geschieht durch die Hinterlegung von erweiterten Funktionszuordnungsdiagrammen hinter

die, im Geschäftsprozess verwendeten Funktionen. Dabei wird eine „high-level“ Sichtweise auf den technischen Ablauf gegeben, ohne in technische Details wie die Spezifikation von Datentypen abzusteigen. Das in **Abbildung 21a** abgebildete eFZD dient als Beispiel der hier dargestellten Modellierung.



(a) Beispiel eines eFZDs (b) Beispiel des generierten Anwendungssystem Diagramms

Abbildung 21: Die aus diesem Prozessschritt entstanden Diagrammtypen

Als Ergebnis dieses Modellierungsschrittes entsteht während der Definition aller im Unternehmen ablaufenden Geschäftsprozesse ein Repository der IT-Landschaft des Unternehmens. Dieses Repository kann anhand eines so genannten Reports automatisch erzeugt werden, sobald ein Geschäftsprozess definiert wurde. Jedes Anwendungssystem muss über alle Modelle hinweg konsistent bleiben. Wird ein Name geändert, oder eine neue Version eingeführt, so muss klar sein, in welchen Prozessen das Anwendungssystem Verwendung findet. Das ARIS Toolset bietet hierzu die Möglichkeit der Wiederverwendung, da in den einzelnen Modellen (EPK oder eFZD) nur so genannte Ausprägungskopien (Objektinstanzen) verwendet werden. Das bedeutet, dass eine etwaige Änderung eines mehrmals verwendeten Anwendungssystems im Repository sich durch alle Modelle „durchpropagiert“. Ein Beispiel für ein Anwendungssystem Diagramm ist in **Abbildung 21b** gegeben.

Leider bietet ARIS im Moment keine Möglichkeit, Anwendungssysteme durch einen Import ihrer Schnittstellendefinitionen (beispielsweise WSDL) zu übernehmen. Auch das Anbinden einer UDDI Komponente, aus der alle im Unternehmen / Unternehmensverbund verwendeten Anwendungen / Services importiert werden können, fehlt.

4.2.3 Komponenten-erstellung/erweiterung

Wenn nach der Modellierung der vorhandenen IT Systeme sowie des Geschäftsprozesses klar wird, dass die gewünschten Funktionalitäten durch die vorhandenen Anwendungssysteme nicht abgedeckt werden können, muss auf eine Erweiterung der vorhandenen, bzw. Neuerstellung von Services/Anwendungssystemen zurückgegriffen werden. Dabei sollte ein Standardmodell der Softwareentwicklung zum Einsatz kommen. Je nach Größe und Anzahl der zu erweiternden/ zu erstellenden Anwendungen bieten sich hier verschiedene

Verfahren an. Dazu zählen die in Kapitel 3 beschriebenen Verfahren des „Spiralmodells“ (Abschnitt 3.4.1), des „Rational Unified Processes“ (Abschnitt 3.4.2) oder des „eXtreme programming“. Im Idealfall sollte das im Unternehmen schon erfolgreich eingesetzte Verfahren zur Softwareentwicklung eingesetzt werden. Dabei können Informationen aus den vorherigen Phasen dieses Vorgehens wichtige Informationen bieten. Dazu gehören beispielsweise die Rahmenbedingungen beziehungsweise der Kontext, in dem das neue Anwendungssystem entsteht sowie Anforderungen an den Geschäftsprozess, die somit auch für eine Komponente gelten müssen.Eine weitere wichtige Fragestellung, die an dieser Stelle noch zu beantworten wäre, ist die des „make or buy“. Wenn Standardlösungen vorhanden sind, die sich mit vertretbarem Aufwand integrieren lassen, sollten die Standardprodukte verwendet werden.

4.2.4 Komponententest

Das Testen der erstellten/erweiterten Komponenten ist untrennbar mit dem vorherigen Schritt verbunden. Falls die Entscheidung zu Gunsten einer Eigenentwicklung fiel, ist ein Testen der neu erstellten Komponenten gegen die Schnittstellen des Prozesses unabdingbar. Anzuraten sind automatisierte Tests, die mithilfe verschiedenster Werkzeuge, die am Markt erhältlich sind, realisiert werden können.Eine weitere Fragestellung die in dieser Phase zu beantworten wäre, ist die Eignung der Komponenten für den Gesamtprozess. Anforderungen die an den Prozess gestellt werden, müssen auch durch seine Teilkomponenten erfüllt werden. So könnte es bei einem Prozess der sehr oft ausgeführt wird und eine festgelegte maximale Antwortzeit erfüllen muss, wichtig sein auch gekaufte Standardprodukte einem Lasttest zu unterziehen.Für weitergehende Informationen bezüglich des Tests von Software siehe zum Beispiel [Mye01].

4.2.5 Komponentenintegration

Die Integration der am Prozess beteiligten Komponenten stellt eines der Hauptaugenmerke dieser Arbeit dar. Wünschenswert wäre eine Vollautomatisierung der Erstellung eines Kompositionsschemas (beispielsweise BPEL) aus den am Anfang des Prozesses erstellten Geschäftsprozessmodellen. Aufgrund der unterschiedlichen Abstraktionsgrade der verschiedenen Modelle ist im Moment eine Vollautomatisierung undenkbar. Die Erweiterung der EPKs um die notwendigen technischen Details sowie Laufzeitaspekte, würde ihrer Intention als intuitive Geschäftsprozessbeschreibungssprache einer fachlichen Sicht nicht gerecht werden. Es ist einem Entscheidungsträger eines Unternehmens nicht zuzumuten, für die Laufzeit wichtige Aspekte bei seiner Modellierung von Geschäftsprozessen miteinzubeziehen. Daher ist eine vollautomatische Transformation unrealistsch. Vielmehr wäre es wünschenswert, dass die verschiedenen Benutzergruppen innerhalb des ganzen Prozesses für sie relevante Informationen aus den Modellen ziehen können beziehungsweise bereitgestellt bekommen.

Um die Aktualität der Informationen zu gewährleisten, ist eine teilautomatisierte Bereitstellung sowie eine Propagierung von Änderungen (Changemanagement) unverzichtbar.

Da die bis zu diesem Prozessschritt durchgeführten Modelle und Änderungen alle auf ein und derselben Datenbasis ablaufen (in dieser Vorgehensweise die ARIS Datenbank mit ihren verschiedenen Diagrammtypen (EPK, eFZD, Anwendungssystem Diagramm, Organigramm etc.)), ist der kritischste Schritt der Übergang zwischen dem fachlich-inhaltlichen Modell und dem technischen Modell.

Da, wie oben schon erwähnt, der Grad der (technischen) Detaillierung innerhalb eines Geschäftsprozessmodells, das anhand der Integration Engineering Methodik erstellt wurde, nicht ausreicht, um eine vollautomatisierte Überführung in ein vollständiges, lauffähiges technisches Modell zu ermöglichen, sollte wenigstens eine teilautomatisierte Variante zur Verfügung gestellt werden. Aus einem EPK Modell müssen demnach alle für eine IT-technische Umsetzung relevanten Informationen zu einem Workflowgerüst extrahiert werden. Dieser gedankliche Ansatz der Transformation bzw. dessen Referenzarchitektur ist als Vorläufer für eine technische Umsetzung zu sehen, wie er in [SV05] vorgeschlagen wird. Der Schwerpunkt dieser Arbeit liegt dabei auf der Konzeption der Transformation die schemanhaft in **Abbildung 22** dargestellt ist. Eine ausführliche Erläuterung der Bedingungen und Regeln, die für eine Transformation in dieser Arbeit erstellt wurden, wird in Kapitel 4.3 gegeben.

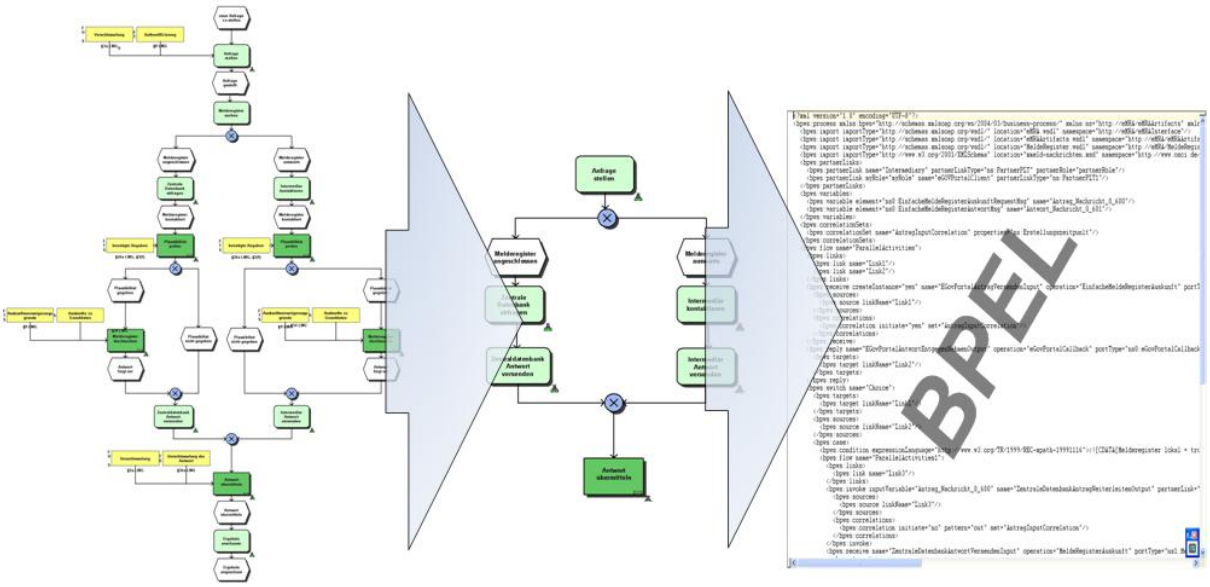


Abbildung 22: Schemenhafte Darstellung des Transformationsalgorithmus

4.2.6 Integrationstest

Um die Qualität der Integrationslösung sicherzustellen, ist ein durchgängiges Qualitätsmanagement vonnöten. Dazu sind nach der Prozessdefinition einige Ziele bezüglich des Gesamtprozesses zu definieren. Blackboxtests, die das Erreichen dieser Ziele sicherstellen, sollten so früh wie möglich definiert und in dieser Phase gegen die Lösung getestet werden. Die meisten EAI Werkzeuge bieten die Möglichkeit, anhand automatisch erstellter

Testclients den kompletten Ablauf eines Geschäftsprozesses zu testen. Der Implementierungsaufwand in diesen Fällen fällt relativ gering aus. Da es im Bereich Qualiätssicherung sehr gut anwendbare Literatur (beispielsweise [Mye01], [AGB03]) gibt, und das Testen ebenfalls nicht zu den Schwerpunkten dieser Arbeit zählt, wird davon abgesehen, diese Problematik zu detailliert zu beschreiben.

4.2.7 Betrieb

Die Übergabe der Gesamtlösung in den Produktivbetrieb sollte erst nach Erreichen der in der vorherigen Phase definierten Ziele durchgeführt werden. Bevor dies der Fall sein kann, müssen die Verantwortlichkeiten für die im Prozess durchgeführten Teilschritte sowie des Gesamtprozesses festgelegt sein. Dies bedeutet, dass zusätzlich zu den Administratoren der im Gesamtprozess genutzten Anwendungssysteme, Prozessadministratoren festgelegt werden müssen. Diese sind zu benachrichtigen, falls es zu einer Fehlersituation im Prozess kommt. Fragestellungen wie die Qualifikation und Schulung der am Prozess beteiligten Mitarbeiter werden jedoch ebenfalls nicht weiter vertieft.

4.2.8 Monitoring/Benchmarking

Das Monitoring stellt eine wichtige Grundlage für die Identifikation von Optimierungsbedarf dar. Dazu ist eine Analyse der durch den Betrieb angefallenen Prozessdaten notwendig. Jedes Integrations- / Workflowprodukt bietet die Möglichkeit, detaillierte Daten über die abgearbeiteten Prozessinstanzen zu sammeln. Diese Daten aus den so genannten Audit trails sollten kontinuierlich analysiert werden. Die aus diesen Daten gewonnen Einsichten können bei der Planung neuer sowie bei der Optimierung der bestehenden Geschäftsprozesse sehr hilfreich sein. So kann anhand einfacher Datamining Verfahren herausgefunden werden, ob es im Prozess zu Engpässen bei der Ausführung kommt und somit eine Aufstockung der Ressourcen nötig ist. Beispiele für Erkenntnisse, die aus diesen Daten gezogen werden sollten, sind:

- Maximale Ausführungszeit des Prozesses
- Minimale Ausführungszeit des Prozesses
- Durchschnittliche Ausführungszeit des Prozesses
- Maximales Datenaufkommen in einem definierten Zeitintervall
- Minimales Datenaufkommen in einem definierten Zeitintervall
- Durchschnittliche Fehlerrate
- Gesamtanzahl der ausgeführten Prozesse

4.2.9 Optimierung

Die aus dem vorigen Prozessschritt gewonnenen Erkenntnisse können für eine kritische Analyse des Geschäftsprozesses herangezogen werden. Die Optimierung des Geschäftsprozesses sollte als eine dauerhafte Aufgabe über den kompletten „Lebenszyklus“ eines Prozesses sein. Dabei sollten die identifizierten Optimierungspotentiale in jedem Aspekt des Geschäftsprozesses ausgeschöpft werden. Dies beinhaltet nicht nur die technische Optimierung, sondern unter Umständen auch eine organisatorische Umstellung. Eine Problematik, die sich in diesem Prozessschritt in Bezug auf die Vorgehensweise ergibt, ist die Retransformation von dem technischen Modell zurück zum fachlich-inhaltlichen Modell. Da über Werkzeuggrenzen hinweg gearbeitet wird, ist eine Retransformation stark problembehaftet. Selbst bei etablierten Verfahren, wie beispielsweise dem UML „round-trip“ engineering, gibt es Probleme bei der korrekten Darstellung von Änderungen auf unterster Modellebene. Da der Schwerpunkt dieser Arbeit auf der Transformation von der inhaltlich-fachlichen Sicht in die technische liegt, kann auf dieses komplexe Themengebiet nicht tiefer eingegangen werden. Diese Problematik konnte im Integration Engineering Projekt, das durch den BMBF gefördert wurde, auch nicht gelöst werden. Auf diesen Aspekt wird noch einmal im Rahmen des Ausblicks und der Zusammenfassung eingegangen (s. Kapitel 6).

4.3 Modelltransformation

Ausgangspunkt für die Modelltransformation in einen technischen Workflow sind ereignisgesteuerte Prozessketten, die anhand der „Integration Engineering“ Methodik erstellt wurden.Dazu gehören auch die den EPKs hinterlegten eFZDs, die in Kapitel 3.1.2 beschrieben wurden. Im Rahmen dieser Arbeit sollte die verwandte Methodik analysiert und in Kapitel 5 anhand eines Praxisbeispiels umgesetzt werden. In diesem Zuge sind einige Schwachstellen identifiziert worden, die in dieser Arbeit durch eine Erweiterung der Methodik beseitigt werden, siehe Abschnitt 4.3.6 dieses Kapitels.

Ein schwerwiegendes Problem, das bei der Transformation auftritt, ist das Fehlen bestimmter Restriktionen an die Modellierung von EPKs, die deren Transformierbarkeit nach BPEL sicher stellen würden. BPEL entstand als Kombination einer „Graphenbeschreibungssprache“ (WSFL [Ley01]) die vorschreibt, dass die mit ihrer Hilfe erstellten Kontrollflusskonstrukte „DAGs“ (directed acyclic graphs) sind sowie einer blockorientierten „sequentiellen“ Sprache Microsofts namens XLANG [Tha01].

EPKs fordern diese Einschränkung nicht. Daher wäre es grundsätzlich möglich, in EPKs „Graphen“ zu modellieren , die Zyklen beinhalten. Zyklen, die Schleifen darstellen, das heißt deren Wiederholungen durch eine Abbruchbedingung eingeschränkt sind und keine Sprünge in bzw. aus ihrem Schleifenrumpf erlauben, sind auch durch BPEL beschreibbar. Zyklen wie sie in EPKs erstellt werden können, weisen diese Einschränkungen jedoch nicht auf. Eine Transformation nach BPEL wäre in diesem Fall unmöglich. Eine detaillierte Diskussion dieser Problematik findet sich in Abschnitt 4.3.3 dieses Kapitels. Um dieses Scheitern der Transformation auszuschließen, wird vorausgesetzt, dass die modellierten Prozesse anhand eines Algorithmuses auf (erweiterte) Zyklenfreiheit untersucht worden sind.

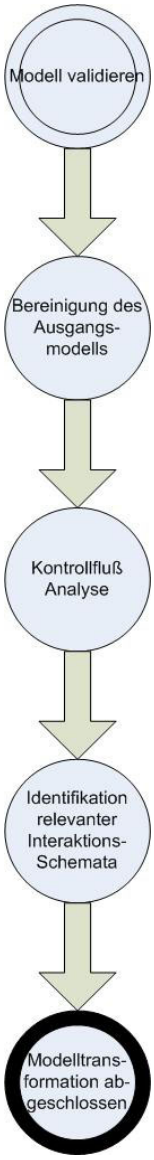
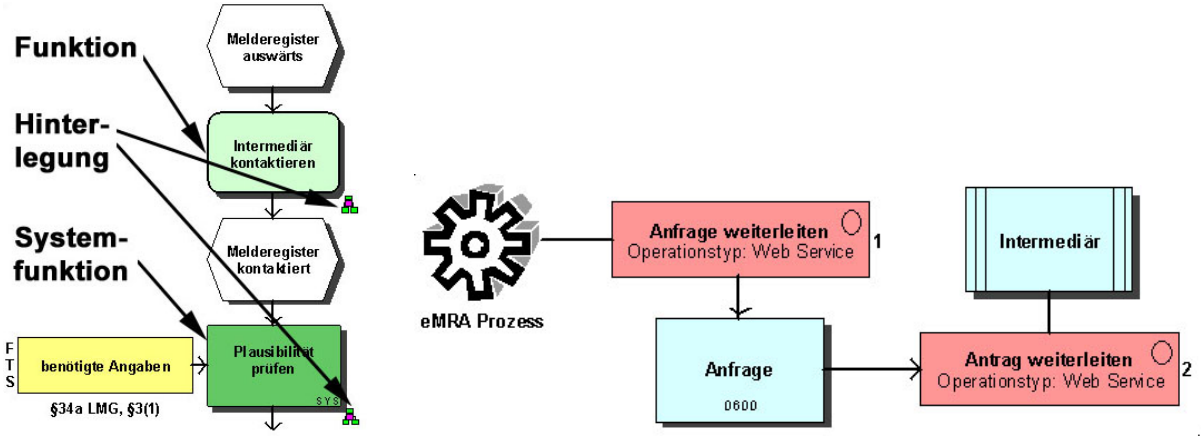


Abbildung 23: Der Transformationsalgorithmus grafisch dargestellt

Ein wichtiges Merkmal, das durch eine Transformation der EPK in ein BPEL Modell nicht verletzt werden sollte, ist, dass der Ausgangsprozess in seinem Zielmodell erkennbar bleiben soll. Um dies zu gewährleisten, sollte jede Funktion, die mit einem eFZD hinterlegt ist, im Normalfall nur jeweils eine Web Service Interaktion realisieren. Somit würde jede Funktion auf genau eine Aktivität abgebildet werden. Die einzelnen Schritte des vorgeschlagenen Algorithmuses (s. **Abbildung 23**) wird in den Unterkapiteln näher erläutert.

Zuerst müssen die für eine Verwendung im technischen Modell relevanten Informationen aus den in Kapitel 3.1.1 beschriebenen EPKs und eFZDs identifiziert werden. BPEL beschreibt ein Kompositionsschema, durch das „Services“ miteinander in einen zeitlichen Zusammenhang gebracht werden können. Aus zustandslosen Web Service Komponenten werden somit zustandsbehaftete Kompositionen. Der Fokus liegt dabei auf der Definition der einzelnen im Prozess verwendeten Partner sowie des Kontroll- und Datenflusses, der zwischen den Prozesspartnern erfolgt. Da in

einer EPK oder eFZD keine konkreten technischen Aussagen über die verwendeten Datentypen gemacht werden, kann bei einer Transformation der Modelle nur ein abstrakter Hinweis auf die Datenstrukturen gegeben werden, zum Beispiel in Form eines Nachrichtentyps.
Auf der Ebene der EPKs sind die zu betrachtenden Modellbausteine die der `Funktionen` und `Systemfunktionen`, die mit einem `eFZD` hinterlegt sind und mindestens eine `Operation` vom Typ *Web Service* beinhalten. **Abbildung 24** stellt die relevanten Bausteine grafisch dar. Andere Funktionen sind entweder manuelle Aktivitäten, die demnach keinen Integrationsbedarf aufweisen oder proprietäre Systeme, die keine direkte Schnittstelle für eine Integration anbieten.



(a) Ausschnitt aus einer EPK (b) Der Funktion „Intermediär kontaktieren“ hinterlegtes eFZD

Abbildung 24: Beispielhafte Darstellung relevanter Modellbausteine

Da die technisch relevanten Informationen für eine Integration in der verwendeten Notation auf eFZDs „ausgelagert“ sind, besteht der für eine Transformation „nutzbare“ Informationsgehalt der EPKs darin, den `Kontrollfluss` zu steuern. In Abschnitt 4.3.1 wird dieser Aspekt der Transformation ausführlich beschrieben.

4.3.1 Anforderungen an den Kontrollfluss

Wie eingangs schon erwähnt, fordert die Modellierung von ereignisgesteuerten Prozessketten kaum Einschränkungen an die modellierten Prozesse. Da jedoch die Zielsprache BPEL sehr klare Restriktionen aufweist, kann nicht jeder Prozess transformiert werden, ohne dass seine Struktur substantiell geändert werden müsste.

Ereignisgesteuerte Prozessketten müssen immer mit mindestens einem Ereignis beginnen und enden. In dieser Arbeit werden sie als Initialereignisse beziehungsweise als Finalereignisse bezeichnet. Jeder durch ein EPK Modell modellierte Graph ist zusammenhängend. Falls in einem Modell mehrere nicht zusammenhängende Graphen modelliert sind, ist

dies ein Modellierungsfehler, da kein Zusammenhang zwischen den einzelnen Prozessen besteht und daher jeder der Graphen in einem separaten Modell modelliert werden sollte. Weiterhin existiert eine Regel für EPKs (s. Kapitel 3.1.1), die besagt, dass es in einem Prozessmodell keine Objekte geben darf, die weder durch eingehende noch ausgehende Kanten verbunden sind. Es wird daher festgelegt, dass jedes EPK Modell genau eine Zusammenhangskomponente haben darf.

Um zu präzisieren, welche Voraussetzungen bei einem Knoten gelten müssen, werden im Folgenden einige Definitionen wie sie in [SGL96] beschrieben sind, vorgestellt. Diese werden später in diesem Abschnitt benötigt und anhand von **Abbildung 25** nachfolgend beschrieben.

DEFINITION ***x dom y :***
Ein Knoten `x` dominiert `y`, genau dann, wenn jeder Pfad vom Start bis zum Knoten `y` immer über `x` führt.

DEFINITION ***x strdom y :***
Ein Knoten `x` dominiert einen Knoten `y` *strikt*, genau dann, wenn x *dom* y und $x \neq y$ gilt.

DEFINITION ***y = idom(x) :***
Ein Knoten `y` ist direkter Dominator eines Knotens `x` , wenn gilt, dass y *strdom* x und es keinen anderen Knoten `z` gibt, so das gelten würde y *strdom* z *strdom* x.

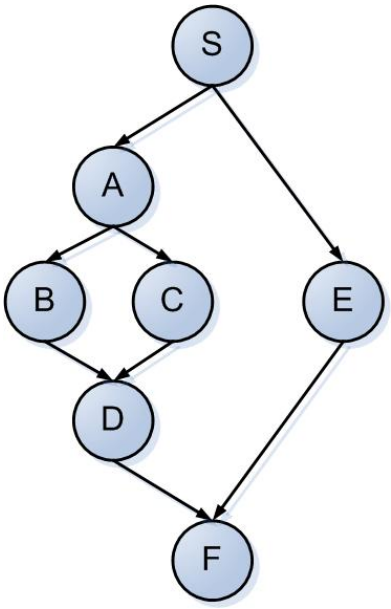
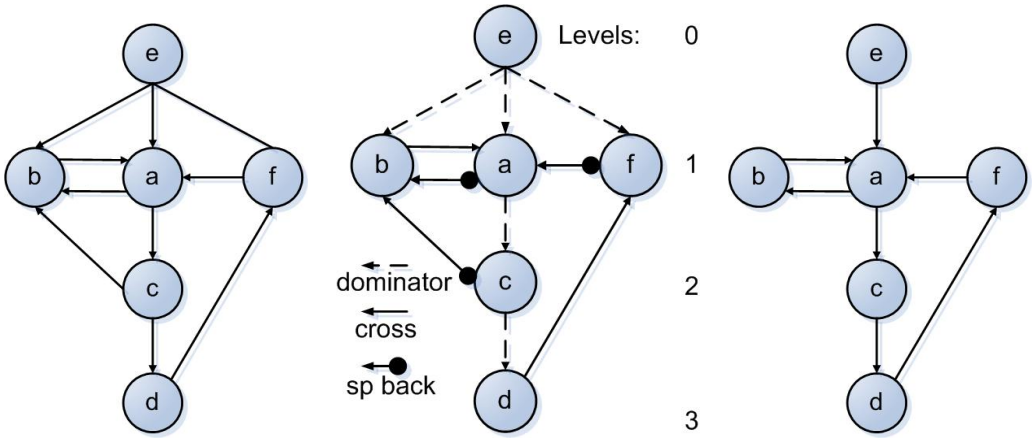


Abbildung 25: Beispiel für die aufgeführten Definitionen

Abbildung 25 zeigt einen beispielhaften Kontrollflussgraphen, anhand dessen die Definitionen verdeutlicht werden. Knoten S dominiert alle anderen Knoten: `S dom {A,B,C,D,E,F}`. Knoten S dominiert alle Knoten strikt `S strdom {A,B,C,D,E,F}` und Knoten A dominiert Knoten B, C und D strikt `A strdom {B,C,D}`. Knoten S ist direkter Dominator der Knoten A und E `S idom({A,E})` . Knoten A ist direkter Dominator der Knoten B und C: `A idom({B,C})`. Grundvoraussetzung für eine Transformation ist, dass anhand EPKs modellierte Modelle von Geschäftsprozessen keine Zyklen beinhalten, die mehrere Eingänge besitzen. Auf diese Problematik wird im Folgenden kurz eingegangen. Eine ausführliche Diskussion der Transformation von Schleifen/Zyklen erfolgt in Abschnitt 4.3.3.4.
Um Zyklen erfolgreich zu erkennen, gibt es verschiedene Algorithmen, die aus der Kontrollflussanalyse des Compilerbaus bekannt sind. Dazu zählen beispielsweise der „backedge“ oder „interval-finding“ Algorithmus, der in [BDL+86] beschrieben wird sowie ein Algorithmus, der mithilfe von so genannten DJ-Graphen sogar „genestete“ Zyklen finden kann [SGL96]. Der erste dieser Algorithmen erkennt jedoch nur Zyklen/Schleifen mit genau einem Einstiegspunkt. Zyklen oder Schleifen, die diesem Kriterium entsprechen, werden, auf Tarjan ([Tar73]) basierend, als „reduzierbare“ Kontrollflussregionen bezeichnet (s. **Abbildung 26c**).

Definition ***reduzierbar :***
Ein Kontrollflussgraph wird in dieser Arbeit als reduzierbar bezeichnet, wenn er nur Schleifen/Zyklen enthält, die genau einen Eingang besitzen. `Eingangsknoten strdom Alle Knoten des Zykluses / der Schleife` .

Schleifen mit mehreren Eingängen werden in der Kontrollflussanalyse nicht reduzierbare („irreducible“) Kontrollflussregionen genannt (s. [HU72]). Da der in [Tar73] und [BDL+86] vorgestellte Algorithmus nur „reduzierbare“, also Schleifen mit nur einem Eingang erkennt, kann er, um ein erstelltes EPK Modell als transformierbar zu validieren, nicht herangezogen werden. Stattdessen sollte der Algorithmus (DJ-Graph), der in dem Artikel von [UM02]/[SGL96] vorgestellt wird, adaptiert werden. Dabei muss dieser nur herangezogen werden, um etwaige Schleifenkonstrukte mit mehreren Eingängen zu erkennen. Nach einer Erkennung eines solchen Konstruktes muss von einer Transformation abgesehen werden, da Schleifen mit mehreren Eingängen, bei einer Transformation in Schleifen mit genau einem Eingang, dazu führen würden, dass das Ausgangsmodell in dem transformierten Modell nicht mehr wieder zu erkennen wäre. Dies rührt daher, dass sowohl bei dem Verfahren des „node-splitting“ als auch bei dem Verfahren, das sich der „DJ-Graphen“ bedient, eine Vervielfältigung der Knoten durchgeführt werden muss. In **Abbildung 26** ist ein Kontrollflussgraph und sein korrespondierender DJ-Graph zu sehen. Die Vor- und Nachteile der beiden Algorithmen werden in [UM02] erörtert.



(a) nicht reduzierbarer Graph (b) zu (a) korrespondierender DJ Graph [UM02] (c) reduzierbarer Graph

Abbildung 26: Beispielgraphen

4.3.2 1. Schritt: Bereinigung des Ausgangsmodells

Der erste Schritt einer Transformation besteht darin, das nach der „Integration Engineering“ Methode entwickelte EPK Modell von allen, für eine Transformation irrelevanten Modellbausteinen zu bereinigen. Alle Funktionen, die nicht durch ein eFZD hinterlegt sind, das mindestens eine Operation vom Typ Web Service beinhaltet sowie die direkt auf sie folgenden Ereignisse werden entfernt. Dies sollte in einem iterativen Verfahren, beginnend vom Startknoten aus, erfolgen. Zuerst werden die Initialereignisse sowie ihre ausgehenden Kanten gelöscht. Daraus ergibt sich, dass die EPK nun mit einer oder mehreren Funktionen ohne eingehende Kanten beginnt. Jede dieser „Initialfunktionen“ wird nun daraufhin überprüft, ob sie mit einem eFZD hinterlegt ist, die eine Operation vom Typ „Web Service“ besitzt. Ist dies nicht der Fall, so wird sie mitsamt dem darauf folgenden Ereignis sowie den zugehörigen Kanten gelöscht. Zu diesem Zeitpunkt kann frühestens ein Operator folgen. Ist dies der Fall, so wird der „Typ“ des Operators geprüft. Wenn es sich um eine Verzweigung („split“) handelt, so werden die darauf folgenden Ereignisse beibehalten. Ist der Operator vom „Typ“ Vereinigung („join“), so wird das darauf folgende Ereignis gelöscht und die eingehende Kante des Ereignisses auf die nachfolgende Funktion gerichtet.

Etwaige Implikationen, die sich daraus ergeben und für eine Validierung verwendet werden können, sind im Kapitel 6 aufgeführt. Die Vorgängerfunktion wird als Ausgangspunkt für die nachzuziehende Kante gewählt, und das Ziel der Kante ist der nächste logische Operator oder die nächste Funktion, die ein eFZD hinterlegt hat, das mit einer WS Operation ausgestattet ist. Da Ereignisse und Funktionen wie in Kapitel 3.1.1 beschrieben, nur Ziel und Ausgangspunkt einer Kante sein können, ist diese „Bereinigung“ des Modells einfach iterativ von dem den Geschäftsprozess auslösenden „Initialereigniss“ aus möglich. Die Struktur des daraus entstehenden Prozesses ist für eine technische Realisierung se-

mantikerhaltend. Aus dieser „Abstraktion“ entsteht ein Modell, das hier Modell 1. Stufe genannt wird. In diesem Modell kommen nur noch folgende Modellbausteine vor.

- Ereignisse, die direkt nach „splits“ auftreten, da sie Informationen beinhalten können, die für die Formulierung von Bedingungen der logischen Operatoren hilfreich sein können.
- Funktionen und Systemfunktionen, die mit einem eFZD hinterlegt sind, das mindestens eine Funktion vom Typ Web Service beinhaltet.
- Logische Operatoren, da sie den Kontrollfluss maßgeblich steuern.
- Kontrollflusskanten, da sie die Ausführungsreihenfolge sowie den Kontrollfluss beschreiben.

Ereignisse stellen nur Daten- bzw. fachliche Zustandsänderungen dar und können vernachlässigt werden. Lediglich Ereignisse nach „splits“ können natürlichsprachliche Bedingungen enthalten, die dem Systemintegrator hilfreich bei der Formulierung von Bedingungen sein könnten. Ein Beispiel für solche Bedingungen wird in **Abbildung 27** gegeben.

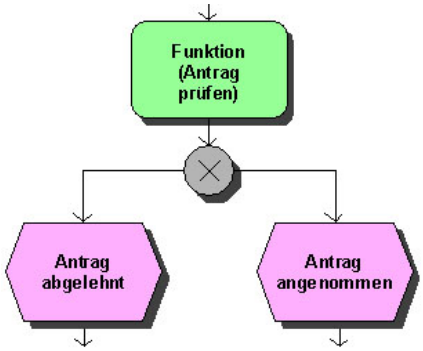


Abbildung 27: Beispiel für Bedingungen in Ereignissen

Funktionen, die mit eFZDs hinterlegt sind, die mindestens eine Operation vom Typ Web Services beinhalten, stellen die wichtigsten Informationen für eine Umsetzung in ein technisches Modell bereit. Dies reicht von den verschiedenen Partnern über die Nachrichten, die ausgetauscht werden, bis hin zu den Operationen, die verwendet werden. Ein Beispiel für relevante Funktionen ist am Anfang des Abschnittes gegeben (**Abbildung 24**).

Logische Operatoren stellen das Kernelement der Kontrollflusssteuerung dar. In der EPK Notation verwendete Operatoren sind OR, XOR sowie AND (s. Abschnitt 3.1.1).

Kontrollflusskanten - Der Kontrollfluss wird aus den EPKs extrahiert. Dabei interessieren vor allem die logischen Operatoren, die EPKs zur Verfügung stellen. Es gibt zwei Ansätze, die Kontrollflusskanten nach BPEL abzubilden. Einerseits anhand des <flow>

Konstruktes (s. Abschnitt 3.3.1), bei dem die Kanten explizit modelliert werden, andererseits anhand von dem Kontrollfluss semantisch äquivalenten BPEL Aktivitäten. In Abschnitt 4.3.3 dieses Kapitels wird auf diese Problematik näher eingegangen. Da die EPKs eine Graphenform besitzen, deren Kanten explizit modelliert sind, liegt eine Abbildung auf das <flow> Konstrukt nahe.

Da bei der Transformation nur Pfade entlang des EPK Modells für eine Integrationslösung relevante Informationen enthalten, auf deren Weg mindestens eine der oben genannten Funktionen vorkommen, dient eine Bereinigung der Übersichtlichkeit. Somit muss der Einstieg in eine Transformation die Identifizierung der ersten erreichbaren relevanten Funktion sein.

Falls der Weg von einem Initialereignis ie bis hin zur ersten relevanten Funktion z, die mit einem eFZD hinterlegt ist, das eine Operation vom Typ Web Service beinhaltet, nur Funktionen und Ereignisse beinhaltet, deren hinterlegte eFZDs keine Operation vom Typ Web Service beinhalten, kann dieser gestrichen werden, da dieser Weg keinerlei Auswirkung auf die Transformation nach BPEL hat. Dies ist auch der Fall, wenn sich auf dem Pfad „splits“ ($s_{1...k}$) befinden, die durch korrespondierende „joins“ ($j_{1...n}$) zusammengeführt werden. Formal bedeutet dies:
`ie strdom s1, s1 strdom jn, k=n,`
Wird der Kontrollfluss bis zur ersten in Betracht kommenden Systemfunktion jedoch geteilt, entstehen daraus Implikationen für die Integration. Diese werden in Kapitel 6 zusammengefasst.

Der Algorithmus zum Bereinigen des Ausgangsmodells wird nachfolgend schemenhaft beschrieben.

```
// Ereignisse erkennen
forEach ( Ereignis e in Graph g )
{
  Object nachfolger = e.getNachfolger();
  Object vorgaenger = e.getVorgaenger();

  if ( (vorgaenger != null) & (vorgaenger == logischerOperator) )
  {
    if ( vorgaenger.ausgehendeKanten.Count > 1 )
    {
      // Der logische Operator ist ein "split"
      // Das Ereignis beschreibt eine Bedingung
      // Das Ereignis muss beibehalten werden
      if ( (nachfolger != null) & (nachfolger == Funktion) )
      {
        // Überprüfen, ob die Funktion ein eFZD hinterlegt hat
        if ( nachfolger.hasEFZD() )
        {
          // Überprüfen, ob das eFZD eine Operation vom Typ WS hat
```

```
          if ( ! nachfolger.getEFZD().hasOperationType("WebService") )
          {
            // Funktion hat keine Op vom Typ WS und wird gelöscht
            e.setAusgehendeKante(nachfolger.getAusgehendeKante());
            g.delete(nachfolger);
          }
        }
      }
    }
    else
    {
     // Der logische Operator stellt einen "join" dar
     // Das Ereignis kann gelöscht werden
     // Wie oben sollte der Nachfolger überprüft werden
     // Falls dieser keine Operation vom Typ WS hinterlegt hat,
     // muss er wie oben gelöscht werden. Die ausgehende Kante des
     // Vorgängers wird auf den Nachfolger des Nachfolgers gerichtet
    }
  }
  // Voränger ist null oder kein logischer Operator
  else
  {
    if ( vorgaenger != null )
    {
      // Vorgänger ist kein logischer Operator
      // Ereignis kann gelöscht werden
      // Wie oben sollte der Nachfolger überprüft werden
      // Falls dieser keine Operation vom Typ WS hinterlegt hat,
      // muss er wie oben gelöscht werden. Die ausgehende Kante des
      // Vorgängers wird auf den Nachfolger des Nachfolgers gerichtet
    }
    else
    {
      // Das Ereignis stellt ein Initialereignis dar
      // es kann gelöscht werden.
      // Wie oben sollte der Nachfolger überprüft werden
      // falls dieser keine Operation vom Typ WS hinterlegt hat,
      // muss er ebenfalls gelöscht werden.
    }
  }
}
```

4.3.3 2. Schritt: Kontrollflussanalyse

Nachdem das Ausgangsmodell als grundsätzlich transformierbar validiert wurde sowie „überflüssige“ Ereignisse und Funktionen entfernt wurden, erfolgt nun die Analyse des abzubildenden Kontrollflusses. In den in diesem Abschnitt gezeigten Schaubildern werden der Vollständigkeit halber jedoch trotzdem noch Ereignisse abgebildet, die nach dem ersten Schritt der Transformation schon entfernt worden wären. BPEL stellt, wie in Abschnitt 3.3.1 spezifiziert, folgende Aktivitäten zur Verfügung, die den Kontrollfluss steuern.

- sequence
- pick
- while
- flow
- switch

Sie wurden in Kapitel 3.3.1 „Strukturierte“ Aktivitäten genannt. Die nach der „Integration Engineering“ Methode entwickelten EPKs und eFZDs vermögen jedoch mehr abzubilden, als durch die obigen Konstrukte in ein technisches Modell umgesetzt werden kann. Die sich daraus ergebenden Einschränkungen und Anforderungen an die Modellierung werden in Abschnitt 4.4 dieses Kapitels erörtert.

4.3.3.1 Linearer Kontrollfluss Die einfachste Art des Kontrollflusses ist eine Aneinanderreihung von Funktionen und Ereignissen ohne jegliche Verzweigung, Vereinigung oder Rückwärtskanten. In **Abbildung 28** wird gezeigt, wie ein solcher Kontrollfluss in einer EPK dargestellt ist. Formal kann diese Art des Kontrollflusses wie folgt charakterisiert werden. Ein Abschnitt linearen Kontrollflusses kann entweder direkt nach dem Initialereignis (IE) oder nach einem logischen Operator (LO) vorkommen und wird immer durch einen logischen Operator (LOE) oder das Ende des Kontrollflusses abgeschlossen. Das bedeutet: Die erste Funktion nach dem Initialereignis oder dem relevanten logischen Operator, in diesem Fall F1 genannt, dominiert alle Funktionen (Fn) auf dem Pfad bis einschließlich der letzten Funktion (Fe), die der direkte Dominator des LOE ist, oder keine ausgehenden Kanten besitzt.
`F1 strdom Fe` $\wedge$ `( (Fe idom LOE)` $\vee$ `(Fe hat keine ausgehenden Kanten) )`
Diese Form des Kontrollflusses lässt grundsätzlich zwei semantisch äquivalente Abbildungen nach BPEL zu.

Abbildung 28: Linearer Kontrollfluss in einer EPK

Einerseits ist es möglich, dieses Konstrukt als <sequence> Konstrukt nach BPEL zu exportieren. Die Reihenfolge der aufgeführten Funktionen würde die Reihenfolge des Auftauchens im <sequence> Element bestimmen. Ein Beispiel hierfür wird im folgenden Codeausschnitt gegeben.

```
... <!-- Umsetzung von Abbildung 28 in ein <sequence> Konstrukt -->
<sequence>
  <invoke name="Funktion1" />
  <invoke name="Funktion2" />
  <invoke name="Funktion3" />
</sequence>
```

Andererseits ist es Möglich, anhand des <flow> Konstruktes in BPEL ein zur <sequence> semantisch äquivalentes Konstrukt zu erstellen. Dabei müssen jedoch zusätzlich zu den Aktivitäten (Funktionen) die Kanten zwischen den Funktionen abgebildet werden. Da keinerlei Bedingungen an die Transitionen zwischen den Funktionen geknüpft sind, kann auf transitionConditions sowie joinConditions (s. Abschnitt 3.3.1) der Aktivitäten verzichtet werden. Im Folgenden wird die Umsetzung des <flow> als BPEL Code aufgeführt. Dabei wird sowohl auf die korrekte Spezifikation der Namespaces als auch auf die vollständige Spezifikation der einzelnen Aktivitäten aus Übersichtlichkeitsgründen verzichtet.

```
... <!-- Umsetzung von Abbildung 28 in ein <flow> Konstrukt -->
<flow>
  <links>
     <link name="Funktion1TOFunktion2" />
     <link name="Funktion2TOFunktion3" />
  </links>
  <invoke name="Funktion1" source="Funktion1TOFunktion2" />
  <invoke name="Funktion2" target="Funktion1TOFunktion2"
                           source="Funktion2TOFunktion3" />
  <invoke name="Funktion3" target="Funktion2TOFunktion3" />
</flow>
```

4.3.3.2 Verzweigungen Eine der wichtigsten Konstrukte sowohl in Kontrollflussdiagrammen als auch in EPKs sind die Verzweigungen. In EPKs werden Verzweigungen durch die logischen Operatoren realisiert. Wie in Kapitel 3.1.1 beschrieben, gibt es drei verschiedene logische Operatoren und somit auch drei Fälle, die zu betrachten sind. Die Diskussion basiert auf dem in **Abbildung 29** modellierten Beispiel für eine Verzweigung. Formal sind sie sehr leicht zu identifizieren. So hat jeder „split“ Operator mindestens zwei ausgehende Kanten. Somit gilt für die ausgehenden Kanten ($KA_{1...n}$):
$|KA_{1...n}| \geq 2$
Bei der Transformation von Verzweigungen und Vereinigungen ist darauf zu achten, das der für eine Verzweigung gewählte Typ die Transformation der korrespondierenden Vereinigung vorschreibt. Diese Fragestellung wird in dem Abschnitt über Vereinigungen (s.

Abschnitt 4.3.3.3) näher beschrieben.
In diesem Beispiel sein alle Funktionen mit einem eFZD hinterlegt, das eine Operation vom Typ „Web Service“ beinhaltet. Wäre dies in einem der Äste nicht der Fall, so würde dieser Teilast nicht transformiert werden. In diesem Falle sollte jedoch eine Warnung ausgegeben werden, da womöglich in dem Teilast „Integrationsbedarf“ herrscht und dieser dann vom Transformationsalgorithmus ignoriert würde. Diese Problematik wird in Kapitel 6 nocheinmal diskutiert.

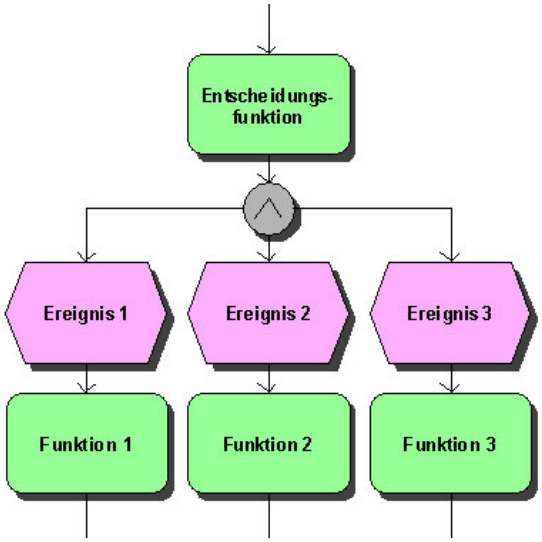


Abbildung 29: Beispiel für eine Verzweigung in einer EPK

Die in **Abbildung 29** benutzte Variante des UND Operators lässt genau eine Abbildung/Transformation nach BPEL zu. Da mit dem UND „split“ Operator ausgedrückt werden soll, dass alle ausgehenden Kanten besucht werden müssen, muss diese Art von Modell in einen <flow> transformiert werden. Das <flow> Konstrukt stellt in BPEL die einzige Möglichkeit dar, die Parallelität der Ausführung zu modellieren.

Wäre der in **Abbildung 29** verwendete Operator ein ODER Operator, so wäre auch nur die Abbildung in einen <flow> möglich, da keine Aussage getroffen werden kann, wie viele der Äste besucht werden würden.

Im Gegensatz zu den letzten beiden Varianten stehen bei der Modellierungsvariante anhand eines XOR Operators zwei Möglichkeiten der Transformation zur Verfügung. Die Semantik dieser Variante schreibt vor, dass genau einer der Äste besucht wird. BPEL bietet in diesem Falle zwei semantisch äquivalente Möglichkeiten einer Abbildung.

Einerseits kann die Transformation wiederum, wie bei den letzten zwei Varianten, anhand des <flow> Konstruktes und geeigneten „transitionConditions“ modelliert werden. BPEL bietet in einem <flow> jedoch nur die Möglichkeit, UND und ODER Konditionen zu verwenden. Um ein logisches exklusives Oder zu modellieren, ist es somit nötig, die Bedingung umzuformen. Im obigen Beispiel (**Abbildung 29**), das mit einem XOR Operator modelliert ist, wäre die zu modellierende „transitionCondition“ der drei vom

Operator ausgehenden links (Ereignis wird mit e abgekürzt):
e1 XOR e2 XOR e3 - das dazu logisch äquivalente Konstrukt nur anhand OR und AND modelliert ist:
(e1 AND NOT e2 AND NOT e3) OR (NOT e1 AND e2 AND NOT e3) OR (NOT e1 AND NOT e2 AND e3)

Andererseits bietet BPEL das <switch> Konstrukt an, das es ebenfalls ermöglicht, genau einen Ast des Kontrollflusses zu verfolgen. Das <switch> Konstrukt selbst ist in Kapitel 3.3.1 beschrieben. Da in diesem Konstrukt immer nur ein „Ast“ (case) durchlaufen werden kann, können die Bedingungen (Ereignisse) direkt übernommen werden. Das folgende Beispiel soll eine Umsetzung verdeutlichen.

```
<switch standard-attributes>
    <case condition="Ereignis1">
      Funktion1 <!-- Platzhalter für korrespondierende Aktivität -->
    </case>
    <case condition="Ereignis2">
      Funktion2 <!-- Platzhalter für korrespondierende Aktivität -->
    </case>
    <case condition="Ereignis3">
      Funktion3 <!-- Platzhalter für korrespondierende Aktivität -->
    </case>
</switch>
```

4.3.3.3 Vereinigungen Ein Geschäftsprozessmodell kann mit beliebig vielen Ereignissen beginnen. Ein Beispiel hierfür ist in **Abbildung 30** gegeben. Wiederum wird hierbei angenommen, dass alle im Diagramm aufgeführten Funktionen in dem hinterlegten eFZD eine Operation vom Typ Web Service beinhalten. Die Semantik dieses Modells kann wie folgt beschrieben werden. Ein Prozess reagiert auf verschiedene Ereignisse. Sobald genau eines dieser Ereignisse eintritt, wird entweder Folge-Funktion1 oder Folge-Funktion2 oder Folge-Funktion3 ausgeführt. Um ein solches Konstrukt zu erkennen, wird der erste vorkommende Operator, im Falle der Vereinigung der „join“ Operator (JO) überprüft. Es darf keinen Knoten (K) auf den verschiedenen Pfaden zu dem relevanten Operator geben für den gilt:
`K strdom JO`
In BPEL gibt es zu diesem Konstrukt ein äquivalentes Element Namens <pick>. Dabei wartet der Prozess, bis er durch das Eintreffen genau einer Nachricht aus einem Set von definierten Nachrichten „instanziert“ wird. Bei der Transformation in eine <pick> Aktivität, die Prozesse instanzieren kann (createInstance=“yes“), ist darauf zu achten, dass keine so genannten „onAlarm“ Handler benutzt werden. Diese Problematik wird in Kapitel 3.3.1 behandelt. Für jede Nachricht, die in den, den Funktionen hinterlegten eFZDs, an eine Operation vom Typ Web Service gesendet wird, muss ein „onMessage„ Handler erstellt werden.

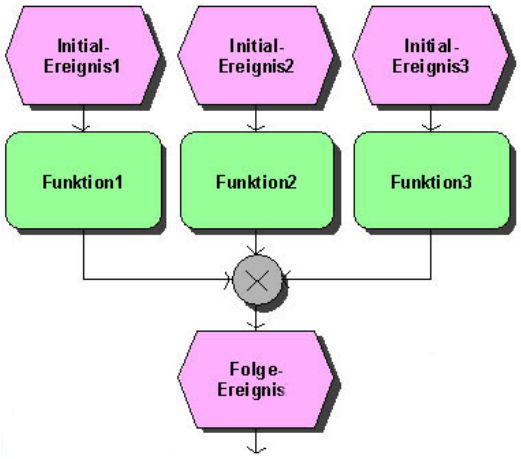


Abbildung 30: Mehrere mögliche Initialereignisse

Wäre das in **Abbildung 30** gezeigte Modell anhand eines UND Operators modelliert worden, müsste dieser Ausschnitt anhand des <flow> Konstruktes transformiert werden. Die Semantik der Darstellung würde ein zwingendes Eintreten aller Initialereignisse fordern. Dies müsste anhand des flow Konstruktes sowie einer <empty> Aktivität mit geeigneter „joinCondition“ realisiert werden. Die <empty> Aktivität würde den logischen AND „join„ Knoten simulieren. Würde das oben dargestellte Modell mit einem OR Operator modelliert, könnte dies nicht nach BPEL exportiert werden, da nie entschieden werden könnte, ob alle möglichen Äste schon durchlaufen worden wären. Daher wird festgelegt, dass bei einer Modellierung mehrerer Initialereignisse kein ODER Operator verwendet werden darf.

Die oben beschriebenen Fälle der Umsetzung in BPEL Konstrukte gelten jedoch nur am Anfang eines Prozesses. Tritt eine Vereinigung später im Prozessverlauf, das heißt nach einem „split“ Operator auf, so richtet sich ihre Transformation nach dem zuvor verwendeten Konstrukt, das im vorigen Abschnitt (4.3.3.2) verwendet wurde. Dies kann wie Folgend beschrieben formal ausgedrückt werden. Jeder „join“ Operator hat mindestens zwei eingehende Kanten. Somit gilt für die eingehenden Kanten ($KE_{1...n}$):
$|KE_{1...n}| \geq 2$
Der zu diesem „join“ Operator (JO) gehörende „split“ Operator (SO), kann einfach über folgende Bedingung identifiziert werden. Der SO ist der erste Knoten (K) auf dem Pfad von JO zum Initialereignis (Traversierung der Kanten in umgekehrter Reihenfolge), für den gilt:
`K strdom JO`
Wie im letzten Abschnitt beschrieben, existieren drei Fälle bei der Betrachtung von Vereinigungen die nicht zum Prozessbeginn vorkommen. Wurde die Verzweigung anhand eines <switch> Konstruktes umgesetzt, ist bei der korrespondierenden Vereinigung nichts zu beachten. Der „join“ Operator kann gestrichen werden, da durch das <switch> Konstrukt sichergestellt wird, dass nur ein Ast durchlaufen wird.
Wurde die zu der Vereinigung gehörende Verzweigung durch eine <flow> Konstrukt abgebildet, sind die restlichen zwei Fälle zu betrachten. Die Umsetzung in einen <flow>

wurde gewählt, wenn das korrespondierende EPK Konstrukt entweder ein OR, XOR oder AND „split“ darstellte. In den Fällen OR und AND kann die Umsetzung des logischen Operators einfach durch ein <empty> Konstrukt umgesetzt werden, das eine entsprechende „joinCondition“ beinhaltet. Da BPEL AND und ODER Bedingungen unterstützt, ist diese Formulierung trivial. BPEL bietet jedoch leider keine Möglichkeit, direkt ein XOR zu verwenden. Demnach muss die joinCondition im XOR Fall logisch umgeformt werden. Somit wird im Falle von drei an einem XOR Operator eingehenden Kontrollflusskanten E1, E2 und E3 die joinCondition zu:
`(E1 AND NOT E2 AND NOT E3) OR (NOT E1 AND E2 AND NOT E3) OR (NOT E1 AND NOT E2 AND E3)`
in BPEL Notation ausgedrückt entspricht dies:

```
(getLinkStatus(E1) AND NOT getLinkStatus(E2) AND NOT getLinkStatus(E3)) OR
(NOT getLinkStatus(E1) AND getLinkStatus(E2) AND NOT getLinkStatus(E3)) OR
(NOT getLinkStatus(E1) AND NOT getLinkStatus(E2) AND getLinkStatus(E3))
```

Man erkennt durch das obige Beispiel leicht, dass alleine die Formulierung dieser Bedingungen in einem BPEL Dokument sehr unübersichtliche Codeabschnitte erzeugen würde. Selbst das obige Beispiel mit nur 3 eingehenden Kanten erzeugt eine Bedingung, die über 3 Bildschirmzeilen reicht. Daher wird festgelegt, dass XOR „split“ und „join“ Operatoren anhand des <switch> Konstruktes abgebildet werden. AND und OR „splits“ und „joins“ werden durch das <flow> Konstrukt transformiert.

4.3.3.4 Schleifen / Zyklen Wie am Anfang diese Kapitels bereits erwähnt, stellen Rücksprünge und die daraus resultierenden Zyklen beziehungsweise Schleifen das schwerwiegendste Problem bei der Abbildung des Kontrollflusses von EPKs nach BPEL dar. Die Erkennung und Lokalisation von Schleifen wird durch den am Anfang dieses Kapitels beschriebenen Algorithmus gewährleistet. Ein Problem in diesem Zusammenhang, auf das in [Kop05] verwiesen wird, ist die Problematik des Oder ($\vee$) Operators. So unterstützen EPKs die Modellierung von Zyklen anhand von Oder Operatoren. Ein Beispiel aus [Kop05] wird in **Abbildung 31** gezeigt.

Aus dieser Modellierung folgt, dass einerseits eine parallele Ausführung von Funktion 1 und Funktion 2 möglich ist. Weiterhin kann Funktion 2 einmal öfter ausgeführt werden, als es Schleifendurchläufe (Bedingung 1, Funktion 1) gibt, obwohl Funktion 2 in keiner Schleife modelliert ist. BPEL erlaubt zwar Parallelität bei der Ausführung, nicht jedoch eine wiederholte Ausführung von Aktivitäten außerhalb eines „while“ Konstruktes. Daher können Zyklen, die mit einem ODER Operator modelliert sind, nicht nach BPEL abgebildet werden. Wird der ODER Operator aus **Abbildung 31** durch einen UND Operator ersetzt, entsteht ein ähnliches Problem. Es ist leicht zu erkennen, dass es zu einer Endlosschleife (Bedingung 1, Funktion 1) kommen würde. Eine Abbildung eines solchen Verhaltens nach BPEL ist nicht möglich. Somit müssen Zyklen immer anhand des XOR (Antivalenz) Operators modelliert werden.

Das wohl schwerwiegendste Problem bei einer Transformation von EPKs nach BPEL sind, wie eben schon beobachtet, die Rückwärtssprünge, die durch EPKs modelliert werden können. Dies ist zum Teil mit den aus der imperativen Programmierung bekannten

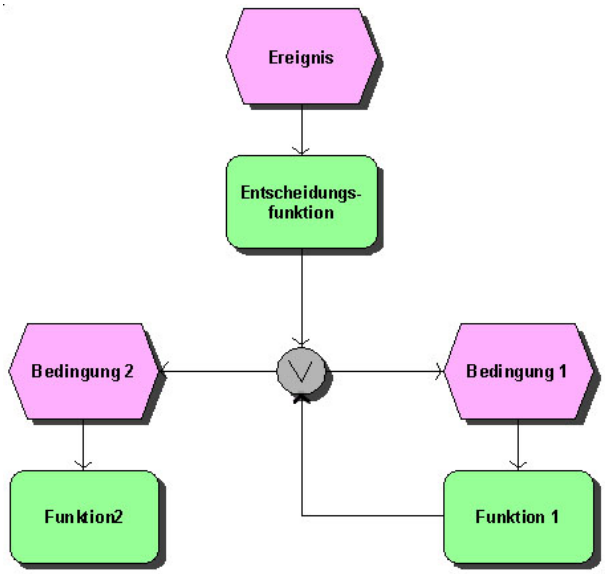


Abbildung 31: Nicht abbildbares Konstrukt [Kop05]

„go-to“ Konstrukten zu vergleichen. Rücksprünge sind immer Ausgangspunkt für Zyklen, wie bereits in **Abbildung 31** zu erkennen war.

Da, wie am Anfang dieses Kapitels geschildert, der Kontrollflussgraph, der durch die EPK modelliert wird, als „reduzierbar“ klassifiziert wurde (vgl. Aschnitt 4.3.1), könnte der Algorithmus von Tajan(intervall/back-edge) angewendet werden, um Schleifen zu erkennen. Der Algorithmus aus [UM02] wurde jedoch schon eingesetzt und adaptiert, um die Transformierbarkeit zu beweisen, und sollte deshalb auch verwendet werden, um Schleifen zu erkennen.

Da alle als „reduzierbar“ eingestuften Modelle nur genau einen Knoten aufweisen, der als Eingang in Betracht kommt, stellt sich nun die Frage nach der inneren Struktur der Schleife sowie ihren Ausgängen.Ein Beispiel einer EPK, die mehrere Ausgänge hat, ist in **Abbildung 32** zu sehen. BPEL bietet zur Modellierung von wiederkehrenden Aktivitäten ein Konstrukt an, das sich „while“ nennt. Charakteristisch für dieses Konstrukt ist, dass es genau einen Eingang und einen Ausgang besitzt. Klassische Schleifen, wie sie aus der Programmierung mit Java oder C-# Sharp bekannt sind, bieten jedoch die Möglichkeit, mehrere Ausgänge zu beschreiben. Dies geschieht mit dem „break“ Operator, der meist nach einer bedingten Verzweigung (if/else Statement) auftritt. Diese Möglichkeit der „erfolgreichen“ frühzeitigen Beendigung einer Schleife bietet BPEL in der aktuellen Version jedoch nicht. In der nächsten geplanten Version von BPEL wird diesem Aspekt durch die Einführung eines „break“ Konstruktes Rechnung getragen ([BPE] Issue No. 142).

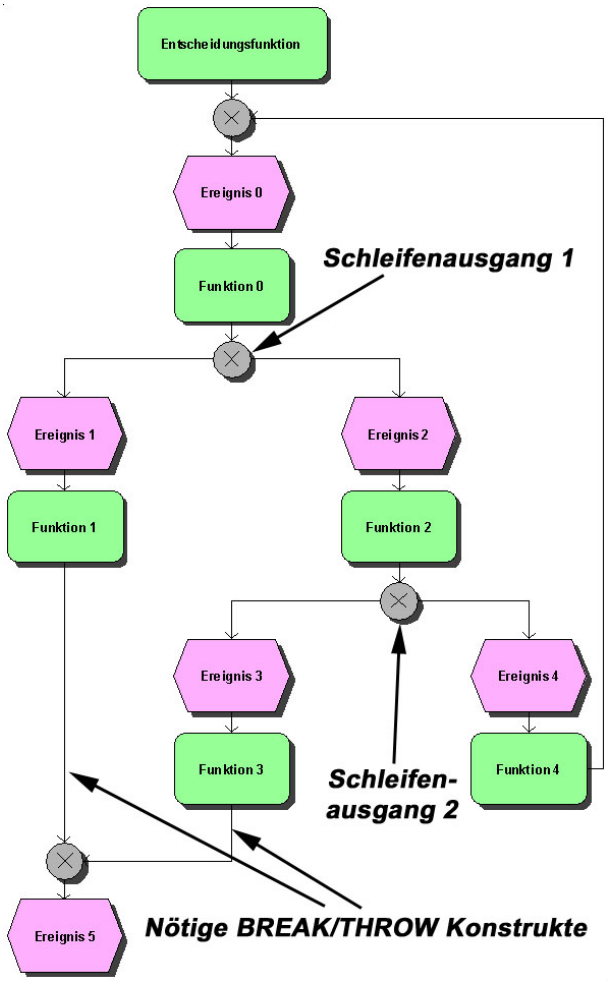


Abbildung 32: Eine EPK, die eine Schleife mit zwei Ausgängen darstellt

Eine Möglichkeit, dieses Verhalten zu emulieren, wäre das „Werfen“ eines Fehlers im BPEL Modell. Da dies jedoch nicht dem Grundsatz entspricht, die Struktur und Semantik des Ausgangsprozesses beizubehalten, sollte von einer Realisierung anhand des „throw“ Konstruktes in BPEL abgesehen werden. Throw impliziert einen Fehler und muss eine Fehlerbehandlung nach sich ziehen. Diese Fehlerbehandlung müsste durch einen leeren „faultHandler“ behandelt werden. Dies wäre jedoch nicht im Sinne der originalen EPK.
Eine andere Möglichkeit wäre die Einführung einer Hilfsvariablen für jeden zusätzlichen Ausgang der Schleife. Diese Variable muss dann mit der Schleifenbedingung mithilfe eines UND Operators verbunden werden. Vor dem Initialisieren der Schleife müsste die Boolsche Variable auf wahr (true) gesetzt werden. Sobald ein Ast, der ein „break“ Konstrukt beinhalten müsste, durchlaufen wird, muss die Variable auf falsch (false) gesetzt werden. Damit wäre dieselbe Funktionalität gegeben, wie sie durch die EPK gefordert wird. Diese Modellierung anhand von Hilfsvariablen wird somit in der in dieser Arbeit vorgeschlagenen Vorgehensweise verwendet.

Nachdem nun sowohl Eingänge als auch Ausgänge von Schleifenkonstrukten klassifiziert und transformiert werden können, muß nun noch kurz auf die innere Struktur der Schlei-

fen eingegangen werden. Grundsätzlich gibt es an die innere Struktur einer Schleife keine Anforderungen, außer die zuvor identifizierten zwei: Eine Schleife kann wiederum andere Schleifen beinhalten, solange diese Schleifen nur einen Eingang haben und mit dem Antivalenz (XOR) Operator modelliert wurden. Der verwendete Algorithmus erkennt und lokalisiert solche genesteten Schleifen. Wie im vorigen Abschnitt erläutert, stellen mehrere Ausgänge auch kein Problem bei einer Transformation dar, da sie mithilfe von Hilfsvariablen modelliert werden können.

Im Folgenden werden die wichtigsten und gebräuchlichsten Schleifentypen, die den oben genannten Charakteristika entsprechen sowie ihre grafischen Entsprechungen in EPKs aufgeführt. Dabei steht die Funktion „Schleifenrumpf“ symbolisch für eine beliebige Verkettung von Funktionen, Ereignissen und Operatoren, solange sie den oben genannten zwei Kriterien entsprechen.

- klassische while Schleife - Bei der Ausführung einer klassischen while Schleife wird zuerst die Schleifenbedingung überprüft. Falls sie zu wahr (true) evaluiert wird, wird der Schleifenrumpf so oft ausgeführt, bis die Bedingung zu falsch evaluiert wird. Falls die Schleifenbedingung vor dem ersten Durchlauf schon zu falsch (false) evaluiert wird, wird der Schleifenrumpf gar nicht ausgeführt.

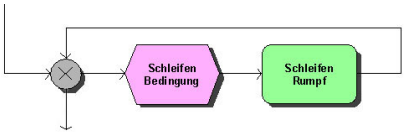


Abbildung 33: Klassische while Schleife

- do-while Schleife - Im Gegensatz zur klassischen while Schleife, wird bei der do-while Schleife der Schleifenrumpf mindestens einmal durchlaufen. Dies passiert auch, wenn die Schleifenbedingung beim ersten Durchlauf schon falsch (false) ergibt.

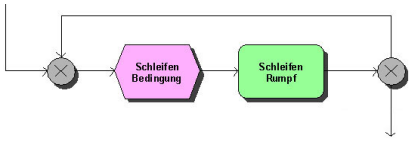


Abbildung 34: Do-while Schleife

- repeat-until Schleife - Bei der repeat-until Schleife wird der Schleifenrumpf immer mindestens einmal durchlaufen, bevor die Schleifenbedingung überprüft wird. Die Besonderheit der repeat-until Schleife zeigt sich in ihrer Schleifenbedingung. Im Gegensatz zu den while Schleifen wird der Schleifenrumpf in repeat-until Schleifen so lange ausgeführt, bis die Schleifenbedingung zu wahr evaluiert wird.

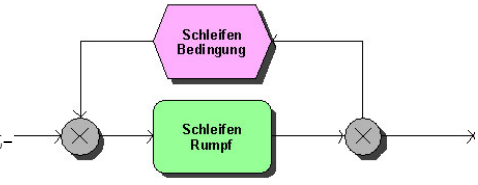


Abbildung 35: Repeat-until Schleife

Alle oben aufgeführten Schleifen haben genau einen „Einstieg“ / Eingang sowie genau einen Ausgang. Somit entsprechen sie den festgelegten Kriterien. Die klassische while Schleife wird einfach anhand des äquivalenten BPEL Konstruktes <while> repräsentiert. Dabei wird der Name des Ereignisses (Schleifenbedingung) als Hinweis für den Bearbeiter des BPEL Gerüstes als Schleifenbedingung übernommen.

Die „do-while“ Schleife wird in BPEL realisiert, indem eine Hilfsvariable eingeführt wird,

die mit der originalen Bedingung „verodert“ wird. Eine Initialisierung dieser Hilfsvariablen mit „true“ (wahr) resultiert darin, dass der Schleifenrumpf mindestens einmal durchlaufen wird. In dem Schleifenrumpf muss die Hilfsvariable dann auf „false“ (falsch) gesetzt werden. Wie bei der klassischen while Schleife wird der Name des Ereignisses (Schleifenbedingung) als Hinweis für den Bearbeiter des BPEL Gerüstes als Schleifenbedingung übernommen. Damit ist das Verhalten einer „do-while“ Schleife emuliert.

Bei der „repeat-until“ Schleife wird, bei einer Transformation nach BPEL, dasselbe Verfahren angewendet wie bei der „do-while“ Schleife. Da in der EPK die Schleifenbedingung schon „negiert“ modelliert wird, muss eine Negation der Schleifenbedingung nicht durchgeführt werden. Repeat-until Schleifen werden ähnlich wie das break Konstrukt in eine neue Version des BPEL Standards mit aufgenommen([BPE] Issue No. 232).

Damit wurde die Problematik von Zyklen und Schleifen im Ausgangsmodell und ihre Entsprechung in BPEL, das <while> Konstrukt, abgearbeitet.

4.3.4 3. Schritt: Identifikation von relevanten Interaktionsschemata

Die im Forschungsprojekt „Integration Engineering“ entwickelte Methodik zur Integration von Anwendungssystemen sollte im Zuge dieser Arbeit auf ihre Praxistauglichkeit hin untersucht/analysiert werden. Da das Hauptaugenmerk dieser Methodik darauf lag, die technischen Aspekte eines Geschäftsprozesses in den, den Funktionen hinterlegten eFZDs zu modellieren, werden in diesem Abschnitt die Schwächen der Methode wohl am deutlichsten sichtbar. An dieser Stelle wird die „Integration Engineering“ Methodik um einige Modellbausteine und Fuktionalitäten erweitert. Veränderungen der Modellierungstechnik werden in den Schaubildern grau hinterlegt und in Kapitel 4.3.6 ausführlich beschrieben.

Die durch ein eFZD beschriebenen Interaktionsschemata werden in BPEL durch folgende Aktivitäten umgesetzt. Diese Schemata sind jedoch nur ein Auszug der durch eFZDs abbildbaren Interaktionen. Für eine Umsetzung in einen BPEL Prozess sind sie jedoch ausreichend. Eine ausführliche Beschreibung der einzelnen BPEL Elemente wird in Kapitel 3.3.1 gegeben.

- <receive>
- <pick>
- <reply>
- asynchrones <invoke>
- synchrones <invoke>

In diesem Abschnitt werden nur die Modelle besprochen, die nach BPEL abgebildet werden können. Es ist beispielsweise auch möglich, Kommunikation zwischen Anwendungssystemen zu modellieren, die nicht „über den BPEL Prozess“ kommunizieren. Das heißt,

sie kommunizieren direkt miteinander. In diesem Fall besteht jedoch kein Handlungsbedarf. Deshalb werden in diesem Abschnitt nur solche Modelltypen behandelt, die auch in ein technisches Modell, das anhand von BPEL beschrieben wird, transformiert werden können.

<***receive***> - Wie in **Abbildung 36** abgebildet, wird das Empfangen einer Nachricht durch den Prozess mittels einer durch den „abstrakten Anwendungssystemtyp“ Prozess zur Verfügung gestellten Operation, die eine Nachricht empfängt, modelliert. In der IE Methodik wurde dies nur implizit dargestellt. Das Anwendungssystem stellte eine Operation zur Verfügung, die eine Nachricht(Entitätstyp) versendete, ohne den Empfänger zu modellieren. In der neu vorgeschlagenen Methodik werden der durch den Prozess zur Verfügung gestellten Methode drei neue Attribute hinterlegt. Das erste Attribut spezifiziert, ob es sich um eine Instanz erstellende Operation handelt. Dies ist immer dann der Fall, wenn die eingehende Nachricht das Initialereignis repräsentiert. Das nächste Attribut spezifiziert, ob die Operation synchron oder asynchron modelliert werden soll. Um eine Identifikation der korrespondierenden Operationen die eine synchrone Kommunikation darstellen, gewährleisten zu können, wurde weiterhin ein Attribut eingeführt, dass einen Identifizierer beinhaltet.

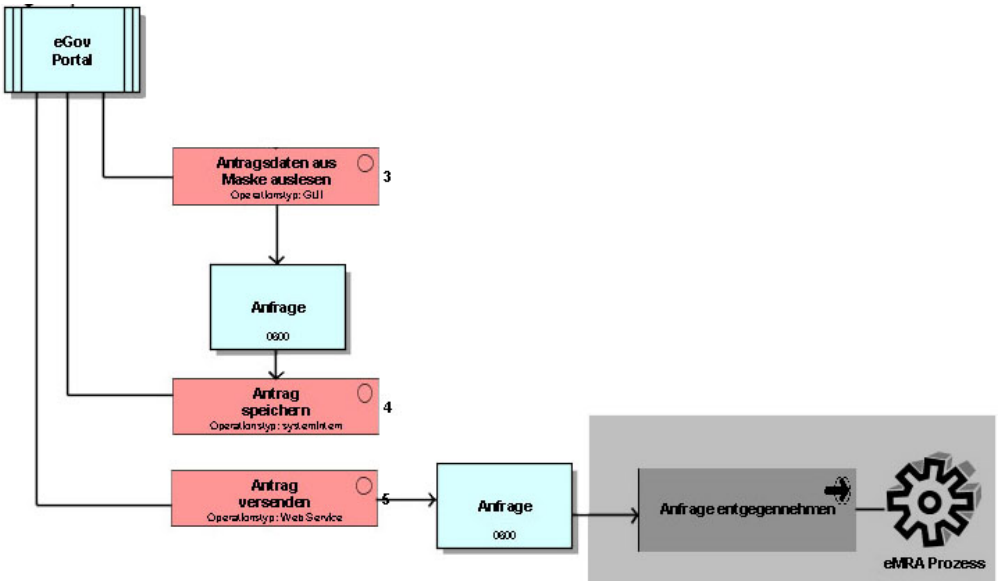


Abbildung 36: Receive Aktivität in einem eFZD

<***pick***> - Ein Unterschied zwischen einer <receive> und einer <pick> Aktivität, innerhalb eines eFZDs, existiert nicht. Dies kann nur anhand der übergeordneten EPK entschieden werden. Ein <pick> wird genau dann modelliert, wenn auf mehrere Initialereignisse parallele Funktionen mit einem eFZD hinterlegt sind, die jeweils ein <receive> modellieren und dann wieder auf Ebene der EPK durch einen XOR Operator zusammengeführt werden. Ein Beispiel hierfür wird in **Abbildung 30** vorgestellt.

<***reply***> - Die reply Aktivität wird in BPEL zusammen mit einer receive Aktivität dazu benutzt, eine nach außen hin synchrone Operation zu modellieren. Reply stellt immer eine Antwort auf eine zuvor durch ein <receive> empfangene Nachricht dar. Falls dieses receive eine initiale Aktivität repräsentiert, die eine Prozessinstanz erstellt, stellt die durch den Prozess nach außen zur Verfügung gestellte Schnittstelle eine synchrone dar. Um die beiden Operationen nachher automatisiert, als zusammengehörig identifizieren zu können, wurde, wie bereits beim <receive> Konstrukt erwähnt, ein Attribut eingeführt, das eine Identifikation gewährleisten kann. Eine synchrone Prozessschnittstelle bereitzustellen bedeutet, dass der den Prozess aufrufende Client so lange wartet, bis eine Antwort durch den Prozess zurückgegeben wird.

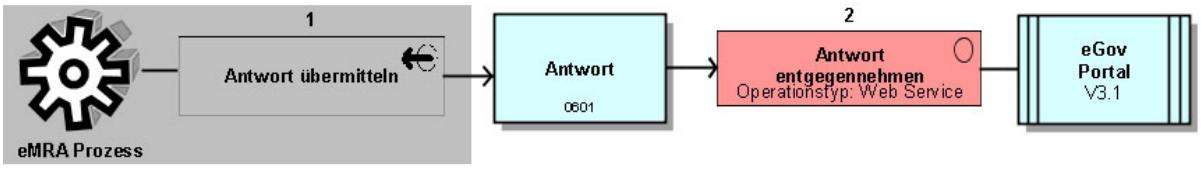


Abbildung 37: Reply in einem eFZD modelliert

asynchrones <***invoke***> - Dieses Konstrukt kann dazu verwendet werden, um Nachrichten an Partner zu schicken. Wird eine Antwort erwartet, kann diese durch eine nachgelagerte receive Aktivität entgegengenommen werden. Das asynchrone invoke wird, wie im Folgenden beschrieben, in eFZDs dargestellt. Der Prozess besitzt, dargestellt durch eine Kante vom Typ „unterstützt“, eine Operation vom Typ „Web Service“. Diese versendet, dargestellt durch den Kantentyp „hat Output“, eine Nachricht (Entitätstyp). Diese Nachricht wird von einer Operation vom Typ „Web Service“ entgegengenommen. Grafisch ist dies dargestellt durch eine Kante vom Typ „ist Input für“. Diese Operation wird durch ein Anwendungssystem zur Verfügung gestellt, was wiederum durch eine Kante des Typs „unterstützt“ grafisch dargestellt wird. **Abbildung 38** stellt ein solches asynchrones invoke dar.

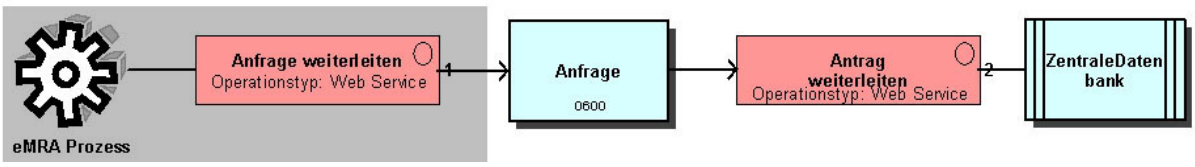


Abbildung 38: Invoke in einem eFZD modelliert

synchrones <***invoke***> - Ein synchrones Invoke differenziert sich dadurch von einem asynchronen invoke, dass der Prozess erwartet, umgehend von der aufgerufenen Operation einen Rückgabewert zu erhalten. Dies bedeutet, dass der Prozess so lange mit der Fortführung seiner Aktivitäten wartet, bis er eine Antwort von seinem Prozesspartner bekommt. Um ein synchrones invoke anhand eines eFZDs darzustellen, muss das asynchrone

invoke um den „Antwortpfad“ erweitert werden. Die vom aufgerufenen Anwendungssystem bereitgestellte Operation vom Typ „Web Service“ hat, zusätzlich zu der eingehenden Kante „ist Input für“, eine ausgehende Kante „hat Output“. Diese Kante führt zu einem Entitätstyp, der die Antwort des Systems auf die initiale Anfrage darstellt. Dieser Entitätstyp ist dann wiederum mit einer Kante vom Typ „ist Input für“ mit der vom Prozess zur Verfügung gestellten Operation verbunden. Ein Beispiel ist in **Abbildung 39** dargestellt.

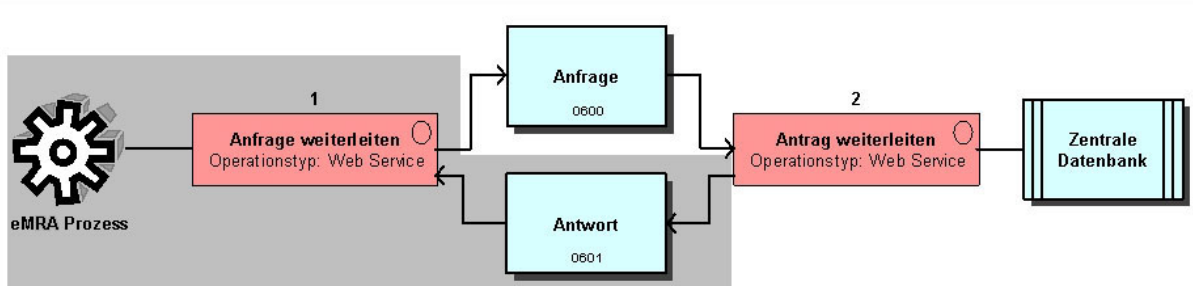


Abbildung 39: Synchrones invoke als eFZD modelliert

4.3.5 Namensgebung

Da Namen innerhalb eines BPEL Prozesses eindeutig sein müssen, werden an dieser Stelle einige Konventionen eingeführt, die eine Zweideutigkeit ausschliessen sollen. Da Leerzeichen und Doppelpunkte in Namen problembehaftet sind, sollten sie bei einer Transformation ersetzt werden. Es wird festgelegt, dass sie durch Unterstriche zu ersetzen sind. Eine andere mögliche Fehlerquelle sind die deutschen Umlaute. Daher wird festgelegt, das sie wie Folgend beschrieben umgesetzt werden: Ä=Ae, Ö=Oe, Ü=Ue.

Die Namen von `Aktivitäten` werden durch die zugrundeliegenden Funktionen bestimmt. Das Schema hierfür lautet demnach: `<EPKFunktionsname>`

`Logische Operatoren` werden durch ihre Entscheidungsfunktionen (direkten Vorgängerfunktionen) sowie ihren Typ gekennzeichnet. Dies gilt jedoch nur bei OR „split“ Operatoren, da sie nur nach Funktionen vorkommen können. XOR wird wie festgelegt immer in ein <switch> Konstrukt transformiert, das den Namen der Entscheidungsfunktion bekommt. Bei und (AND) Operatoren scheint eine Nummerierung auf Basis ihres Vorkommens angebracht. Dies ist dadurch begründet, dass vor AND Operatoren nicht zwangsweise eine Entscheidungsfunktion vorkommen muss. Bei „join“ Operatoren wird anstatt der Entscheidungsfunktion, die direkte Nachfolgerfunktion für die Namensgebung herangezogen. Somit ergeben sich für logische Operatoren folgende Namensschemata:
`OR<split|join><Entscheidungsfunktionsname|FolgeFunktionsname>,`
`AND<split|join><FortlaufendeNummer>`.

`Variablennamen` ergeben sich aus den in einem eFZD spezifizierten Entitätstypen. Dabei werden die Entitätstypen mit dem Zusatz Input, für eine durch den Kantentyp *„ist Input für“* verbundene, oder Output für eine durch den Kantentyp *„hat Output“* verbundene

Kante konkateniert. Damit ergibt sich folgendes Schema:
`<EntitätstypenName>Input`, bzw. `<EntitätstypenName>Output`.

Die Namen der `Services`, die an dem Prozess beteiligt sind, ergeben sich direkt aus den Namen der Anwendungssysteme, die mindestens eine Operation vom Typ Web Service aufweisen. Der Name wird direkt übernommen und mit dem Zusatz Service konkateniert.
`<AnwendungssystemName>Service`

Die Namen von `Links` in einem BPEL (Flow) Kompositionsschema werden in der hier vorgestellten Vorgehensweise wie folgt gebildet. Die Ausgangsfunktion einer Kontrollflusskante dient als erster Namensteil, die Funktion, auf den die Kontrollflusskante zeigt, als zweiter Namensteil. Um die Richtung der Kante natürlichsprachlich zu verdeutlichen, werden die beiden Namensteile mit einem „TO“ verbunden. Daraus ergibt sich folgendes Schema:
`<Ausgangsfunktion>TO<Zielfunktion>`.

Die Benennung von `PortTypes` wird folgendermaßen durchgeführt. Der Name des den Partnerservice repräsentierenden Anwendungssystems wird mit der Endung PT konkateniert. Folgendes Schema gilt:
`<AnwendungssystemName>PT`

Da `Operationen` direkt auch in der vorhandenen Vorgehensweise modelliert werden, wird deren Name direkt für den Namen der Operationen herangezogen.
`<OperationsName>`

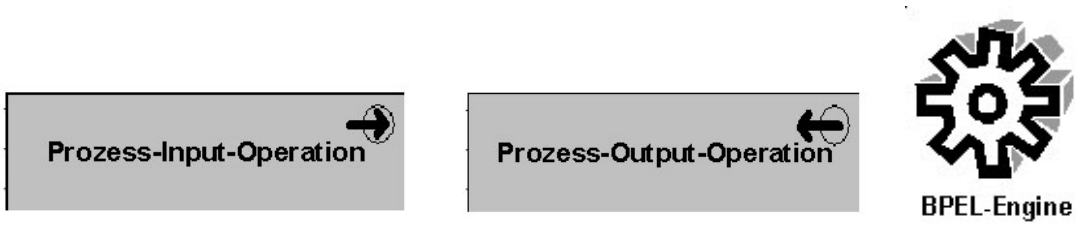
`PartnerLinks` werden werden wie die PortTypes benannt, mit dem kleinen Unterschied, dass anstatt der Endung PT, die Endung PL verwendet wird. Daraus ergibt sich folgendes Schema:
`<AnwendungssystemName>PL`

Wie die Namen der PartnerLinks werden auch die Namen der `partnerLinkTypes` gebildet. Es wird lediglich ein T, für Type, konkateniert.
`<AnwendungssystemName>PLT`

Namen von `Messages` lassen sich direkt von den ausgetauschten Entitätstypen ableiten. Dazu wird der Entitätstypenname mit MSG für Message konkateniert.
`<EntitätstypenName>MSG`

4.3.6 Änderungen an der „Integration Engineering“ Methodik

Im Rahmen der technischen Analyse, die in den letzten Abschnitten beschrieben wurde, offenbarten sich einige Mängel der „Integration Engineering“ Methode, die in einer Erweiterung der Methodik adressiert werden sollten. In der in Abschnitt 3.1.2.1 beschriebenen „IE“ Methodik kann nicht unterschieden werden, ob ein modelliertes Anwendungssystem direkt mit einem anderen Anwendungssystem kommuniziert oder ob diese Kommunikation über den Prozess (die Prozessmaschine) abgewickelt wird. Daher wurde ein neuer Typ von Anwendungssystem vorgeschlagen, der den gerade modellierten Prozess darstellt (s. **Abbildung 40**). Dies beseitigt diese Mehrdeutigkeit und macht eine korrekte Abbildung in ein technisches Modellgerüst möglich.

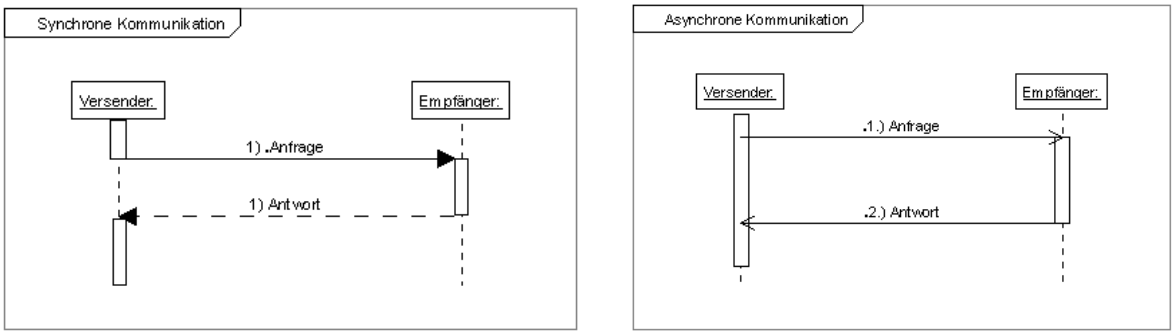


(a) Das Prozess Input Symbol (b) Das Prozess Output Symbol (c) Das Prozess Symbol

Abbildung 40: Die neu eingeführten Symboltypen

Ein weiteres Problem bestand darin, dass es nicht möglich war, maschinell zu entscheiden, ob eine Interaktion mit einem Anwendungssystem synchron oder asynchron umgesetzt werden sollte. Bei der Integration von Anwendungssystemen kann die Kommunikation unter verschiedenen Partnern in zwei verschiedenen Modi ablaufen. Einerseits, wie in **Abbildung 41a** gezeigt, synchron. In diesem Falle wartet der anfragende Dienst mit der Ausführung seines Prozesses so lange, bis er von dem aufgerufenen Dienst eine Antwort bekommt.

Andererseits gibt es den asynchronen Fall (s. **Abbildung 41b**), in dem der aufrufende Dienst mit der Abarbeitung seiner Aufgaben fortfährt, bis er eine Antwort bekommt. In diesem Falle ist eine Korrelation zwischen Anfrage und Antwort unabdingbar, da sonst der Kontext der Kommunikation verloren geht. Zu diesem Zwecke wurden dem Operationstyp, wie er durch die IE Methode in eFZDs verwendet wird, drei neue Attribute hinterlegt. Das erste Attribut zeigt an, ob es sich um eine synchrone oder asynchrone Kommunikation handelt und das zweite Attribut dient dazu, Operationen miteinander über eFZD Grenzen hinweg zu assoziieren. Das zweite Attribut wird notwendig, da es aus fachlicher Sicht nötig sein kann, Funktionen / Operationen zu trennen, während es aus technischer Sicht sinnvoller ist, die Kommunikation synchron umzusetzen. Das letzte Attribut wurde dadurch nötig, da in einigen Fällen klar sein muss, ob es sich um eine Operation handelt, die eine neue Prozessinstanz erstellt, oder nicht.



(a) Synchrone Kommunikation (b) Asynchrone Kommunikation

Abbildung 41: Beispielhafte UML Sequenzdiagramme

BPEL bietet die Möglichkeit, einen Prozess, selbst nach außen hin, entweder synchron oder asynchron darzustellen. Dies geschieht im synchronen Fall durch eine receive/reply Modellierung der Initialaktivität beziehungsweise Endaktivität. Im asynchronen Fall durch eine receive/invoke Modellierung. Da Prozesse in der IE Methodik nach außen hin immer als „Prozessmodul“ dargestellt wurden, wurde nicht klar welche Schnittstellen sie nach außen hin, aus technischer Sicht, zur Verfügung stellten. Dieses Problem wurde im Rahmen dieser Arbeit gelöst indem so genannte „Prozessschnittstellen“ eingeführt wurden. Es gibt zwei verschiedene Arten von Prozessschnittstellen: Die Prozesseingangsschnittstelle (s. **Abbildung 40a**) und die Prozessausgangsschnittstelle (s. **Abbildung 40b**). Wie im obigen Abschnitt beschrieben, können auch diese Schnittstellen „synchron“ oder „asynchron“ modelliert werden. Zu diesem Zweck wurde festgelegt, dass Prozesse, die in anderen Prozessen verwendet werden, als Anwendungssysteme modelliert werden. Der Vorteil dieser Darstellung aus technischer Sicht ist eine Darstellung des Gesamtprozesses nach außen hin als Anwendungssystem, das leicht in andere Prozesse eingebunden werden kann. Weiterhin werden die Schnittstellen, die dieser Prozess bietet deutlicher.

4.4 Anforderungen an die Geschäftsprozessmodellierung

In diesem Abschnitt werden die grundsätzlichen Voraussetzungen an die Geschäftsprozessmodellierung zusammengefasst, die gelten, um ein fachliches Geschäftsprozessmodell in ein technisches Modell zu überführen.
In Kapitel 4.3 wurden folgende Kriterien identifiziert, die eine Überführung eines fachlichen Geschäftsprozessmodells in einen Workflow unmöglich machen:

- Zyklen, die mehrere Eingänge haben
- Zyklen, die durch andere logische Operatoren als den XOR Operator modelliert werden
- Modellierung mehrerer Initialereignisse und Funktionen anhand eines ODER Operators

Zyklen, die mehrere Eingänge haben - In Abschnitt 4.3.1 wurde festgestellt, dass bei der Transformation von Zyklen, die mehrere Eingänge besitzen, ein Verfahren angewendet werden müsste, das sich „node-splitting“ nennt. Dieses Verfahren würde jedoch dazu führen, dass der zugrundeliegende Geschäftsprozess in seinem technischen Modell nicht mehr erkennbar wäre. Deshalb wurde festgelegt, dass nur Modelle transformiert werden, die keine Zyklen mit mehreren Eingängen beinhalten. Ein grafisches Beispiel für einen Zyklus mit mehreren Eingängen ist in **Abbildung 42** gegeben.

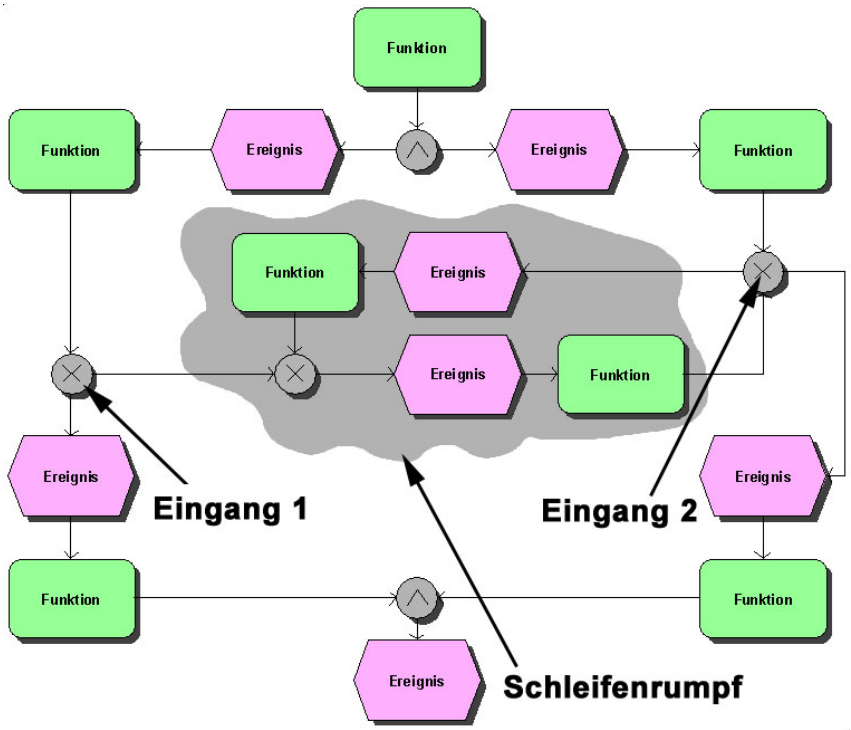


Abbildung 42: Mehrere Eingänge einer Schleife

Zyklen, die durch andere logische Operatoren als den XOR Operator modelliert werden - Die Modellierung von Zyklen anhand der OR und AND Operatoren würde zu einem Nicht-Determinismus des technischen Modells führen und kann in BPEL nicht abgebildet werden. Diese Problematik wurde in Abschnitt 4.3.3.4 identifiziert und erläutert.

Modellierung mehrerer Initialereignisse und Funktionen anhand eines ODER Operators - Ähnlich wie bei dem zuletzt behandelten Problem, kann bei der Modellierung mehrerer Initialereignisse anhand eines ODER „join“ Operators nicht maschinell entschieden werden, ob oder wann alle möglichen Initialereignisse und deren nachfolgende Funktionen eingetreten beziehungsweise abgearbeitet wurden. Dies würde darin resultieren, dass die Ausführung des Prozesses entweder nach dem Eintreten des ersten Initialereignisses fortgeführt würde oder über den OR „join“ Operator nie hinaus käme.

Dies ist nicht erstrebenswert und wird deswegen ausgeschlossen. Daher dürfen mehrere Initialereignisse und die darauf folgenden Funktionen immer nur mittels eines XOR oder AND „join“ Operators modelliert werden. Diese Problematik wurde in Abschnitt 4.3.3.3 behandelt.

Außer den oben genannten drei Punkten, werden keine weiteren Anforderungen an die Geschäftsprozessmodellierung gestellt. Mit dieser Diskussion wird das Kaptiel Vorgehensweise geschlossen. Die Vorgehensweise wird nun im folgenden Kapitel anhand eines Praxisbeispiels beschrieben und umgesetzt.

5 Umsetzung

In diesem Teil der Arbeit wird die in Kapitel 4 vorgestellte Vorgehensweise anhand eines Szenarios aus dem Umfeld des E-Government validiert. Viele Erkentnisse, die zu der Erweiterung der Methodik aus dem Projekt Integration Engineering führten, stammen aus diesem praktischen Anwendungsfall. So wären einige Problematiken bei der Transformation vom fachlichen in das technische Modell wohl nicht so schnell identifiziert worden, wenn sie nicht anhand dieser technischen Realisierung zu Tage gekommen wären.

Das behandelte Szenario kommt aus der Praxis und ist im Rahmen des Forschungsprojektes OrVia in Zusammenarbeit mit dem DVZ Mecklenburg-Vorpommern [dvz] entstanden.

5.1 Ausgangslage

Eine der wichtigsten verwaltungstechnischen Aufgaben des deutschen Staates ist das Führen von Melderegistern, in denen alle Staatsbürger der Bundesrepublik Deutschland aufgelistet sind. In einem demokratischen Staat ist es unabdingbar, einen Überblick über alle Bürger zu besitzen, da ansonsten weder Wahlen ausgerichtet noch Steuern eingetrieben werden könnten. Diese Aufgabe ist im Melderechtsrahmengesetz (MRRG) verankert und in Form von Landesmelderecht in den einzelnen Ländern gesetzlich geregelt. Dabei fällt aufgrund der föderalen Struktur der Bundesrepublik Deutschland die Hoheit über diese Datenhaltung den einzelnen Kommunen zu. Es ist leicht einzusehen, dass dabei über die Jahre eine Vielzahl von proprietären Lösungen sowie kommerziellen Branchenlösungen zum Einsatz kamen. Kleinere Kommunen könnten ihre Melderegister in Form von Microsoft Access Datenbanken implementiert haben, während größere Städte Anwendungssysteme von den Branchenspezialisten erworben haben. Jedoch gibt es zwischen den Herstellern dieser so genannten Fachverfahren keine Standards zum Austausch von relevanten Daten. Im Zuge der Initiative „Deutschland online“ [Bun04] sowie der darauf folgenden Initiative „Bund online 2005“ [Bun05b], wurden in einer Standardisierungsbestrebung exemplarisch, anhand des Meldewesens, die Nachrichten und Inhalte spezifiziert, die innerhalb dieser Domäne ausgetauscht werden. Natürlich speichern die im Moment eingesetzten „Melderegister“ diese Daten auch schon heute. Der Austausch über eine standardisierte Schnittstelle erfolgt jedoch bis dato noch nicht.

Um dem Bürger und der Wirtschaft die Dienste auf bequeme Art und Weise über das Internet zugänglich zu machen, ist ein hoher Integrationsaufwand auf Seiten der Länder und Kommunen zu betreiben. Meist sind im Auftrag der Länder private Gesellschaften öffentlichen Rechts entstanden, die die IT Infrastruktur für Kommunen und Landkreise betreiben.

5.2 Geschäftsprozessmodellierung

Der Prozess der einfachen Melderegisterauskunft (eMRA) wurde mithilfe des ARIS Toolsets von IDS Scheer modelliert. Die ARIS Methodik wurde schon im Forschungsprojekt Integration Engineering um einen sogenannten Methodenfilter erweitert, um die aus diesem Forschungsprojekt entstandene Methodik optimal zu unterstützen. Dieser Methodenfilter wurde nun durch die aus Kapitel 4 resultierenden Neuerungen noch einmal erweitert. In Zusammenarbeit mit der Datenverarbeitungszentrale Mecklenburg-Vorpommern [dvz] und unter Verwendung der, in Abschnitt 2.1.3.2 beschriebenen gesetzlichen Grundlagen, wurde ein Geschäftsprozessmodell erstellt. Dieses Modell, das in **Abbildung 43** zu sehen ist, bildet den Ausgangspunkt für eine Integration der beteiligten Anwendungssysteme. Im folgenden Abschnitt wird der Prozess der eMRA natürlichsprachlich erläutert. Um den Prozess verständlicher zu beschreiben, wird in diesem Abschnitt ein Teil der erst in der nächsten Phase (IT-Modellierung Abschnitt 5.3) zu erstellenden Modelle vorgestellt. Diese Modelle sind mit einem * gekennzeichnet.

Ein Benutzer, dies kann ein privater Bürger oder eine Firma wie beispielsweise ein Inkassounternehmen sein, will Informationen über eine Person einholen. Um an diese Informationen zu kommen, muss der Benutzer sich dem System gegenüber authentifizieren. Dies geschieht um einen Mißbrauch zu verhindern sowie eine Abrechnung zu ermöglichen. Das Landesmeldegesetz Mecklenburg-Vorpommerns sieht vor, dass der Nutzer wenigstens Grundinformationen über die Person besitzt, über die er eine Auskunft einholen will. Im Landesmeldegesetz wird festgelegt, dass der/die Anfragende mindestens drei Angaben zur gesuchten Person machen muss. Dabei zählen Vor- und Nachname als eine Angabe, Geburtsdatum, Geschlecht oder Anschrift ebenfalls. Eine Anschrift besteht aus Postleitzahl, Ort, Straße und Hausnummer. Nachdem der Nutzer seine Anfrage in eine Eingabemaske eingegeben hat, speichert das Portal seine Anfrage und verschickt dann diese an den BPEL Prozess. In **Abbildung 44** werden die Teilschritte, die bis jetzt beschrieben wurden, als eFZD dargestellt. Dieses eFZD ist der EPK der Funktion „`Anfrage stellen`“ hinterlegt.

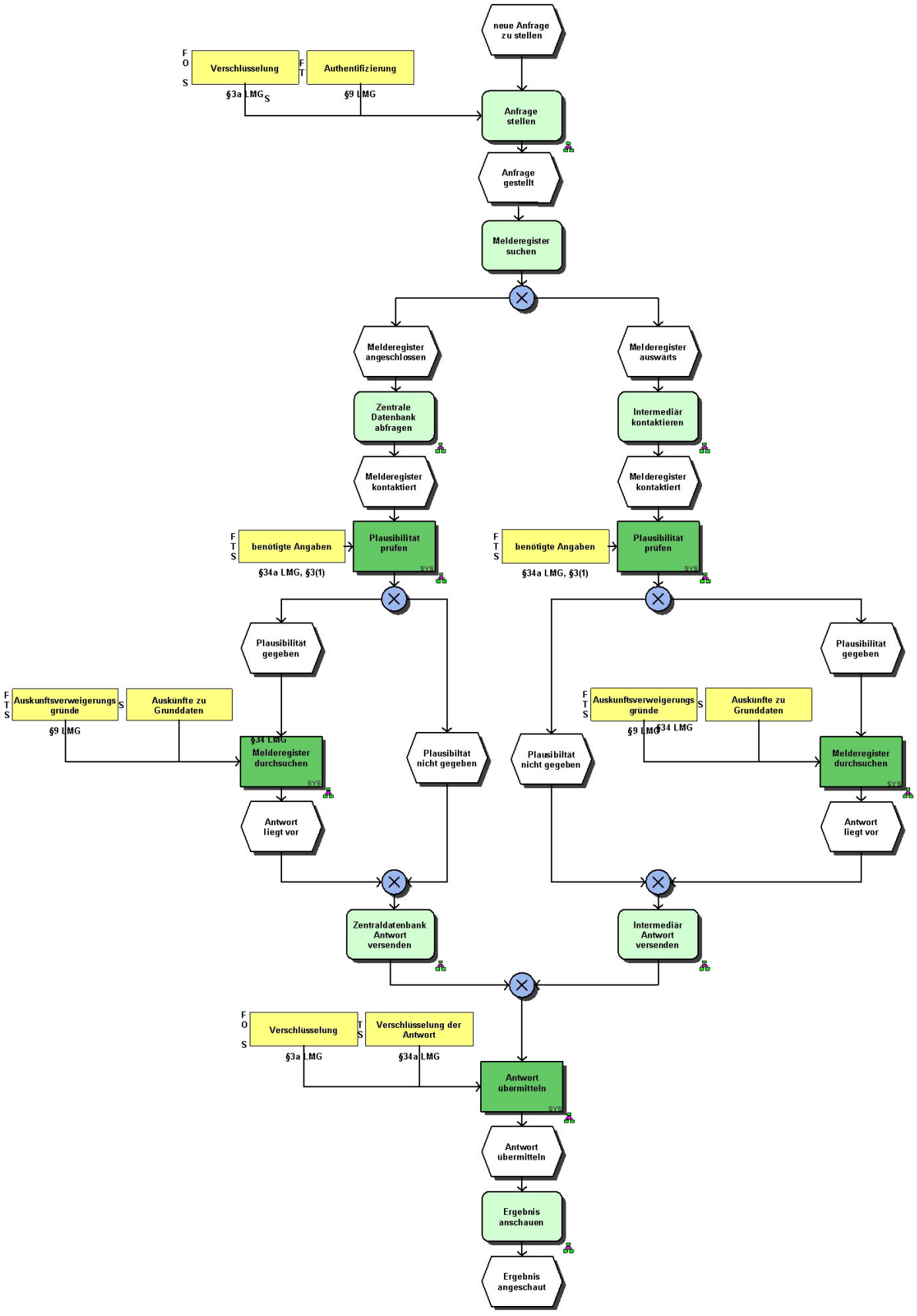


Abbildung 43: Die einfache Melderegisterauskunft als EPK modelliert

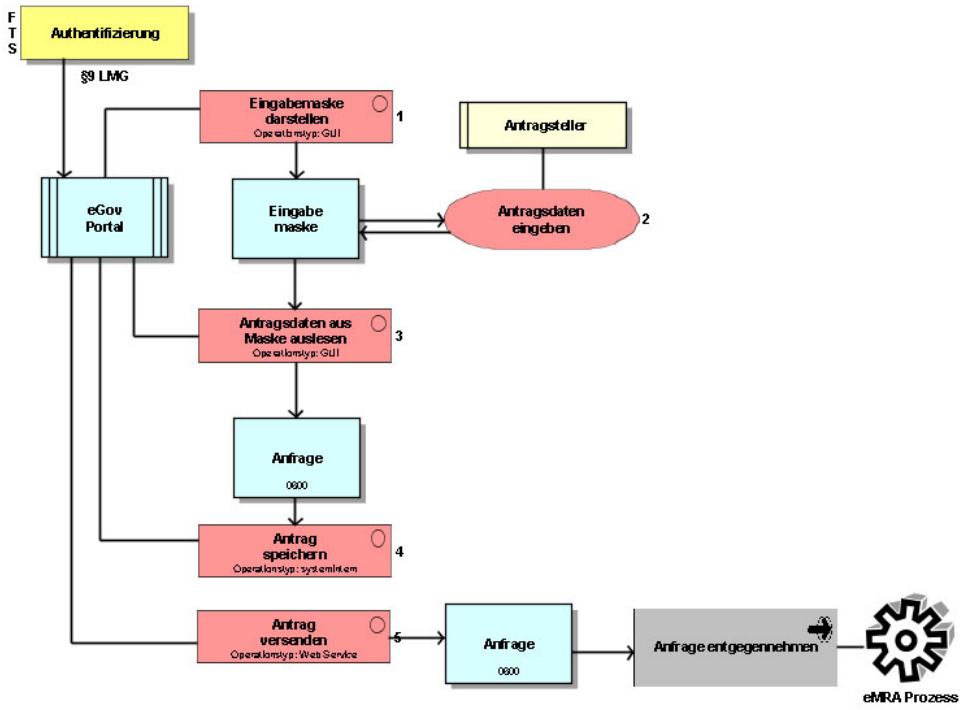


Abbildung 44: „Anfrage stellen“ eFZD *

Als nächster Schritt wird überprüft, ob das Melderegister, an das sich die Anfrage richtet, direkt an das DVZ Mecklenburg-Vorpommern angeschlossen ist(„`Melderegister suchen`“), oder ob es sich beispielsweise um ein Einwohnermeldeamt in Baden-Württemberg handelt. Falls es direkt angeschlossen ist, kann die Anfrage direkt an die zentrale Datenbank weitergeleitet werden („`Zentrale Datenbank abfragen`“), ist dies nicht der Fall, so wird die Anfrage an einen Intermediär geschickt („`Intermediär kontaktieren`“), der die Anfrage an die korrekte Meldebehörde weiterleitet. Da beide Teilschritte bis auf die Namensgebung und die beteiligten Anwendungssystemtypen gleich sind, wird an dieser Stelle nur das eFZD dargestellt, das die Anfrage an die zentrale Datenbank darstellt (**Abbildung 45**).

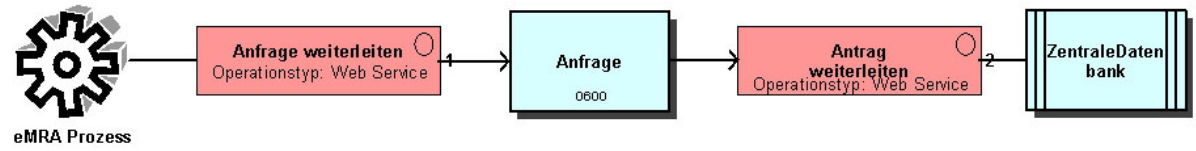


Abbildung 45: „Zentrale Datenbank abfragen“ eFZD *

Nachdem die Anfrage an das Melderegister eingegangen ist, sieht das Gesetz vor, eine Überprüfung der Anfrage auf ihre Plausibilität durchzuführen. Eine Anfrage ist genau dann plausibel, wenn der Anfragende auskunftsberechtigt ist und keine Auskunftssperre hinterlegt ist. Diese Plausibilitätsprüfung wird durch die beteiligten Fachverfahren durchgeführt und wird deswegen als Systemfunktion abgebildet („`Plausibilität prüfen`“).

Das der Systemfunktion „`Plausibilität prüfen`“ hinterlegte eFZD ist in **Abbildung 46** abgebildet.

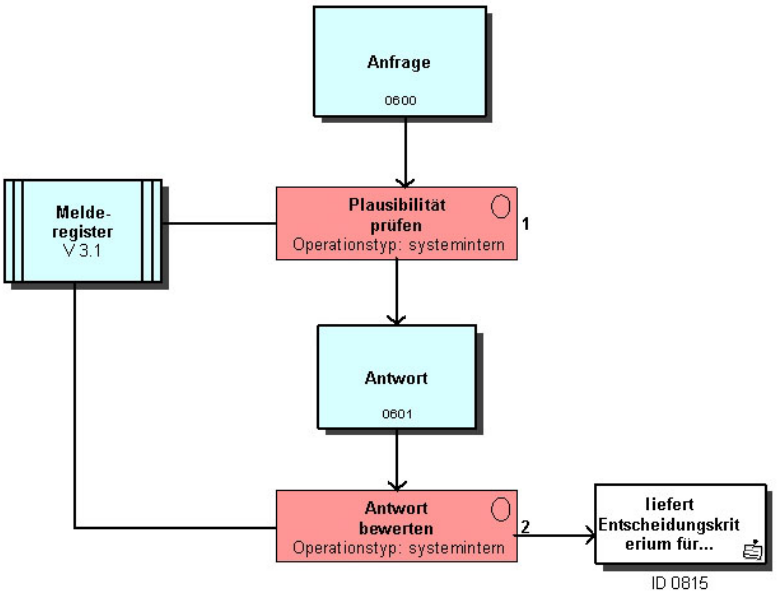


Abbildung 46: „Plausibilität prüfen“ eFZD *

Ist die Plausibilität der Anfrage gegeben, werden in beiden „äquivalenten“ Teilästen der EPK die Melderegister durchsucht und eine Antwort zusammengestellt („`Melderegister durchsuchen`“). Ist eine Plausibilität nicht gegeben, so wird eine Antwort mit einer Fehlermeldung zusammengestellt.

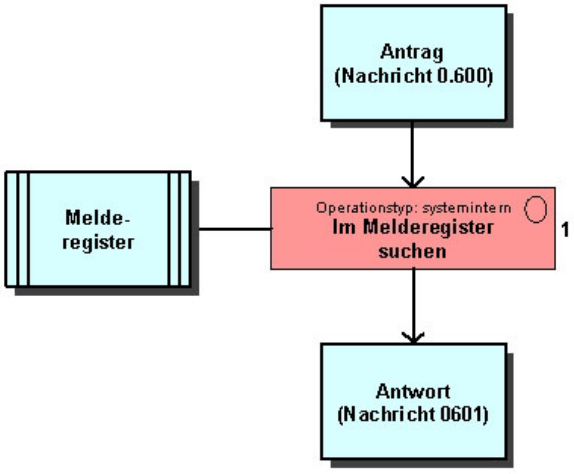


Abbildung 47: „Melderegister durchsuchen“ eFZD *

Nachdem die Antwort erstellt wurde, schickt die Zentrale Datenbank („`Zentraldatenbank Antwort versenden`“), beziehungsweise respektive der Intermediär(„`Intermediär Antwort versenden`“), die Antwort an den BPEL Prozess.

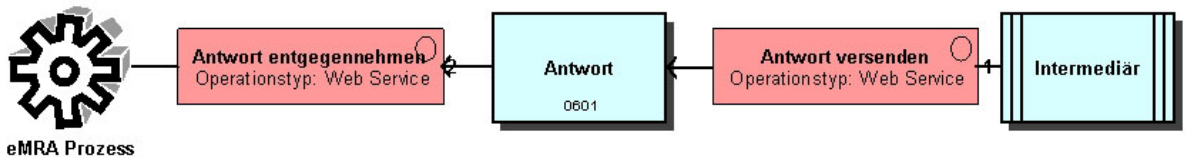


Abbildung 48: „Intermediär Antwort versenden“ eFZD *

Der BPEL Prozess versendet dann wiederum die empfangene Antwort an das E-Government Portal („`Antwort übermitteln`“), wo im letzten Schritt dieser EPK ein Benutzer das Ergebnis der Anfrage betrachten kann („`Ergebnis anschauen`“).

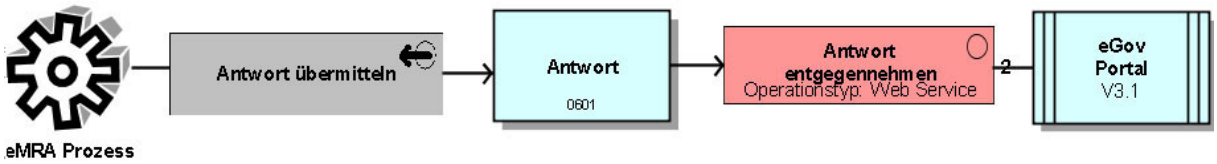


Abbildung 49: „Antwort übermitteln“ eFZD *

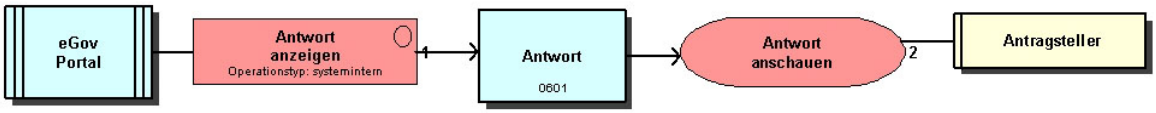


Abbildung 50: „Ergebnis anschauen“ eFZD *

5.3 IT-Modellierung

Nachdem in der letzten Phase des Vorgehens der komplette Prozess der einfachen Melderegisterauskunft fachlich beschrieben wurde, werden nun durch einen IT Spezialisten, der einen Überblick über die bestehenden IT Systeme hat, die fachlichen Funktionen mit eFZDs hinterlegt. Da der Verständlichkeit halber im letzten Abschnitt die eFZDs schon vorgestellt wurden, wird in diesem Abschnitt auf diesen Schritt nicht weiter eingegangen. Nachdem die eFZDs modelliert wurden, stehen die am Geschäftsprozess beteiligten Anwendungssysteme fest. Sie alle werden in einem separaten Modell, dem Anwendungssystem Diagramm zusammengefasst.Wie in Kapitel 4.2.2 beschrieben, soll hierbei ein Überblick über bestehende Anwendungssysteme entstehen. Nachdem sämtliche Geschäftspro-

zesse, die durch IT-Systeme unterstützt werden, modelliert wurden, sollten alle im Unternehmen eingesetzten Anwendungssysteme, in diesem Fall die des DVZ Mecklenburg-Vorpommern, in diesem Diagramm abgebildet sein. Die Anwendungssysteme aus allen modellierten Geschäftsprozessen können in dem ARIS Toolset durch Reports automatisch in ein Diagramm eingefügt werden. Die Kommunikation zwischen den an der fachlichen Modellierung beteiligten Personen und dem Team das die IT-Modellierung durchführt, ist für ein Gelingen der Integration unerlässlich.

Abbildung 51 zeigt das nach dieser Phase entstandene Anwendungssystem Diagramm. Es sollten zwei verschiedene Diagramme erstellt werden, da eine Übersicht über vorhandene Anwendungssysteme wichtig ist, jedoch auch Informationen über die von einem Anwendungssystem angebotenen Operationen relevant sein können. Um ein möglichst übersichtliches Modell zu erhalten, wäre es nicht ratsam, auf nur ein Modell mit allen Informationen zurückzugreifen.

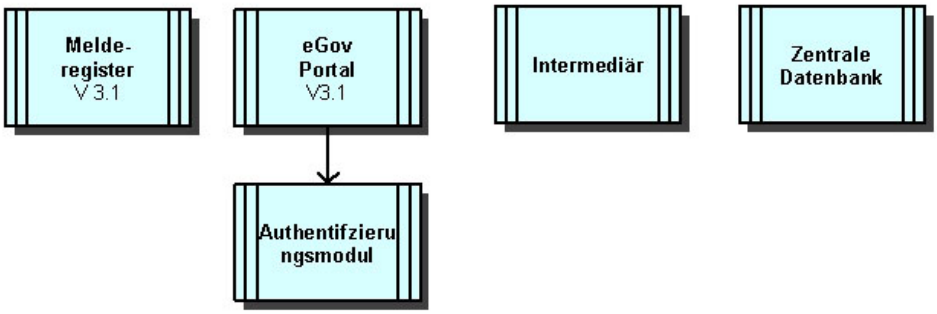


Abbildung 51: Das resultierende Anwendunssystem Diagramm

5.4 Komponentenentwicklung

Da das Anwendungsszenario prototypistisch umgesetzt werden sollte, mussten alle direkt am Prozess beteiligten Systeme nachmodelliert werden. Im Normalfall ist davon auszugehen, dass die Anzahl der neu zu entwickelnden Systeme, im Vergleich zu den involvierten Systemen, sehr gering ist.

Um den BPEL Prozess der nach einer Transformation des fachlichen Geschäftsprozesses entsteht, auszulösen, wurde eine Web-Oberfläche mit dazugehörigem Web Service entwickelt, die das E-Government Portal nachbildet, das die DVZ im Auftrag der Regierung Mecklenburg-Vorpommerns betreibt. Da die ausgetauschten Nachrichten durch den XMeld Standard von vornherein feststanden, wurde das so genannte „contract first developement“ angewandt. Dabei wurde unter Verwendung der Eclipse Plattform aus einer WSDL Schnittstellendefinition ein Web Service Client Stub generiert. Dieser wurde um die nötige Funktionalität erweitert und mit einer Weboberfläche, die über Java Server Faces (JSF) implementiert wurde, versehen. Von einer Implementierung des OSCI Transport Standards wurde aus Komplexitätsgründen abgesehen. In **Abbildung 52** ist die erstellte Webseite zur Anmeldung am Portal abgebildet.

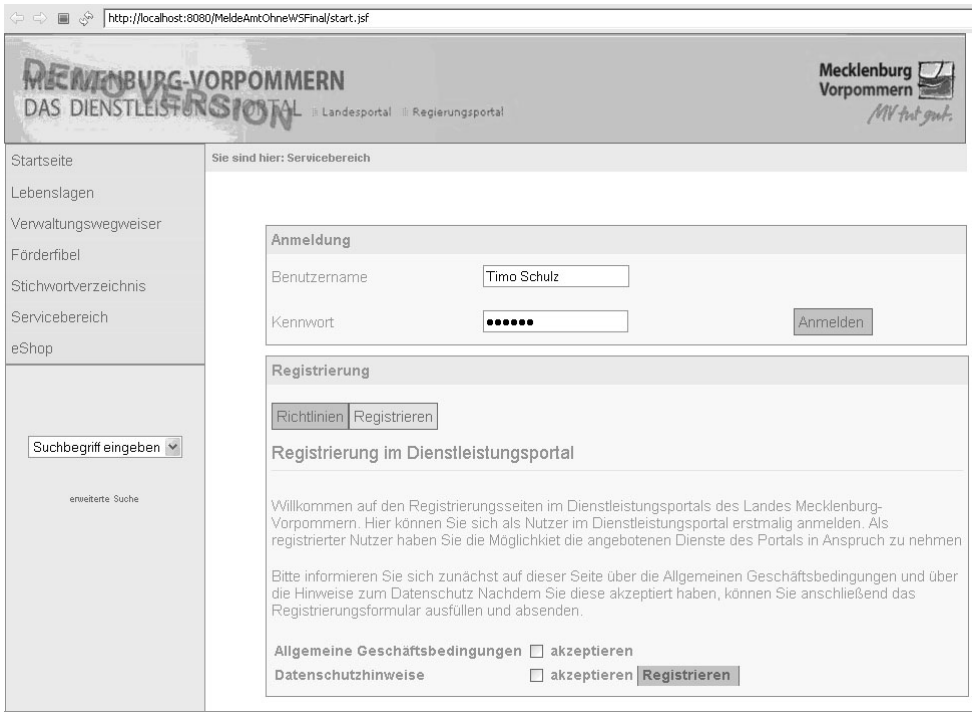


Abbildung 52: Die Weboberfläche des WebServices

Abbildung 53 stellt einen Bildschirmauszug der Anfrage Webseite dar, in der die Daten für die einfache Melderegisterauskunft erfragt werden.

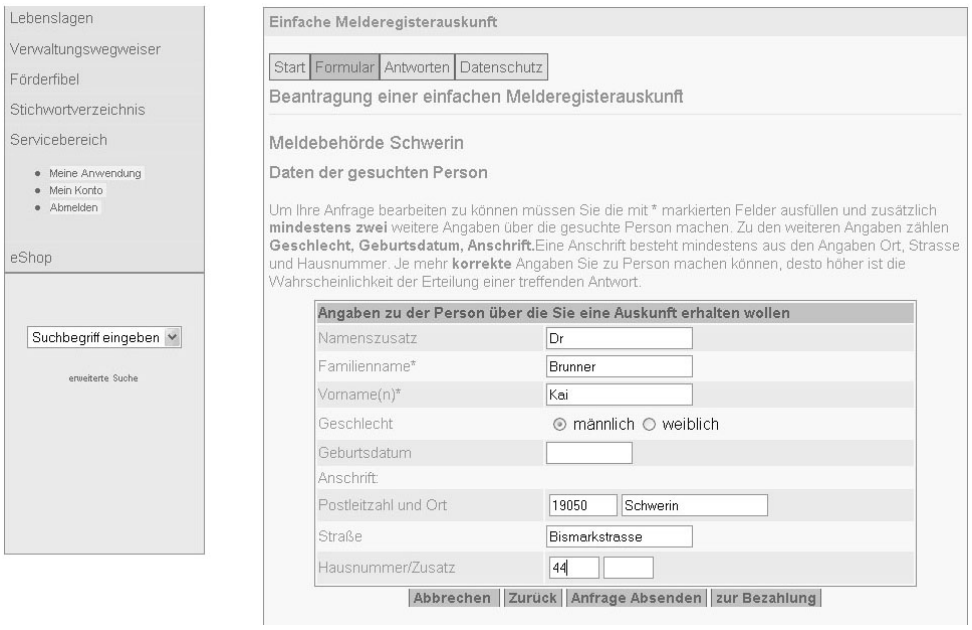


Abbildung 53: Bildschirmabzug „Beantragung einer eMRA“

Die Antwort, die auf eine Anfrage vom Workflowmanagement System bereitgestellt wird, ist in **Abbildung 54** zu sehen.

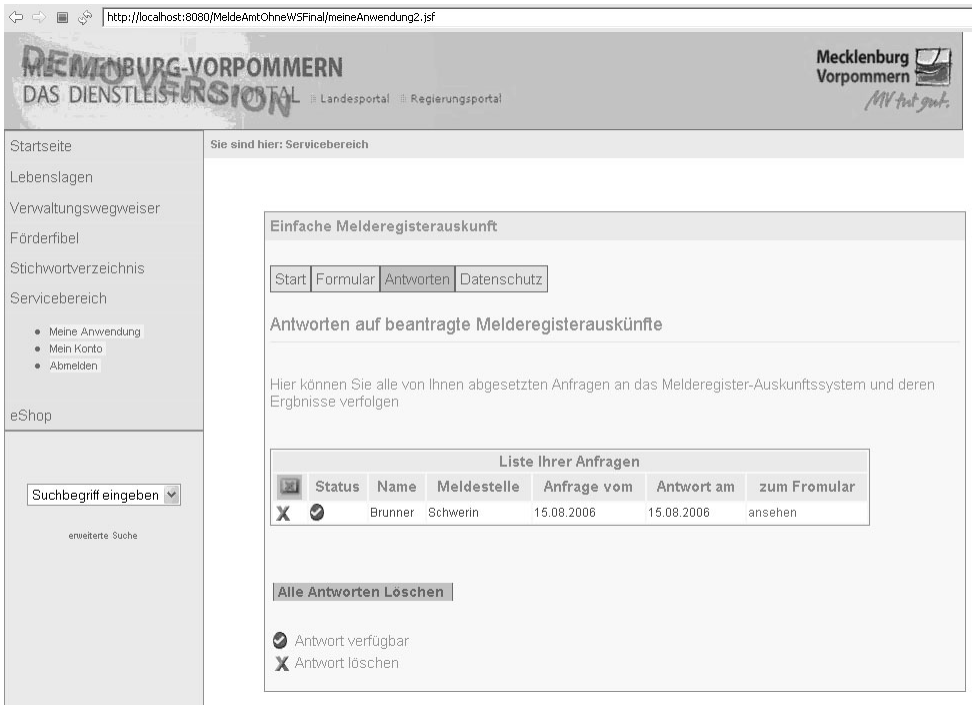


Abbildung 54: Bildschirmabzug „Antwort auf eMRA“

Wie im vorherigen Abschnitt beschrieben, mussten außer dem E-Government Portal noch jeweils ein Web Service für die Zentrale Datenbank sowie für den Intermediär erstellt werden. Wiederum aus Komplexitätsgründen unterscheiden sich die beiden erstellten Services lediglich aufgrund ihres physikalischen Standortes im Netzwerk sowie in der Namensgebung. Die Funktionalität der beiden Services ist beidesmal dieselbe. Eine XMeld Anfrage vom Typ 0600 wird angenommen. Danach wird aus der mittels JDBC angeschlossenen Datenbank eine Antwort vom Typ 0601 zusammengestellt und an den Prozess zurück geschickt.

Da das in **Abbildung 51** angegebene Melderegister keine Web Service Schnittstelle zur Verfügung stellt, konnte von einer prototypistischen Implementierung abgesehen werden. Die Funktionalität wurde in den beiden Web Services Intermediär und Zentrale Datenbank nachempfunden. Damit sind alle an dem Szenario beteiligten Systeme umgesetzt und können nun getestet werden.

5.5 Komponententest

Das Testen der einzelnen entwickelten Komponenten wurde aus Zeitgründen auf ein Minimum reduziert. Für jeden Web Service wurde ein „client“ erstellt, der die empfangenen und gesendeten Nachrichten analysierte.

5.6 Komponentenintegration

In diesem Abschnitt wird die wohl wichtigste Phase dieser Arbeit dargestellt. Zusammen mit den theoretischen Überlegungen aus Kapitel 4.3 wird nun das vorgestellte Verfahren praktisch durchgeführt.

5.6.1 Anforderungen an den Kontrollfluss

Das Ausgangsmodell, das in Kapitel 5.2 beschrieben wurde und in **Abbildung 43** abgebildet ist, beinhaltet keinerlei Zyklen. Daher kann eine Transformation in ein technisches Modell durchgeführt werden.

5.6.2 1. Schritt: Bereinigung des Ausgangsmodells

Nach den in Kapitel 4.3.2 beschriebenen Regeln wurde nun der fachliche Prozess von allen, für die technische Umsetzung irrelevanten Objekttypen bereinigt. Das „virtuelle“ Modell würde nun wie in **Abbildung 55** gezeigt aussehen.

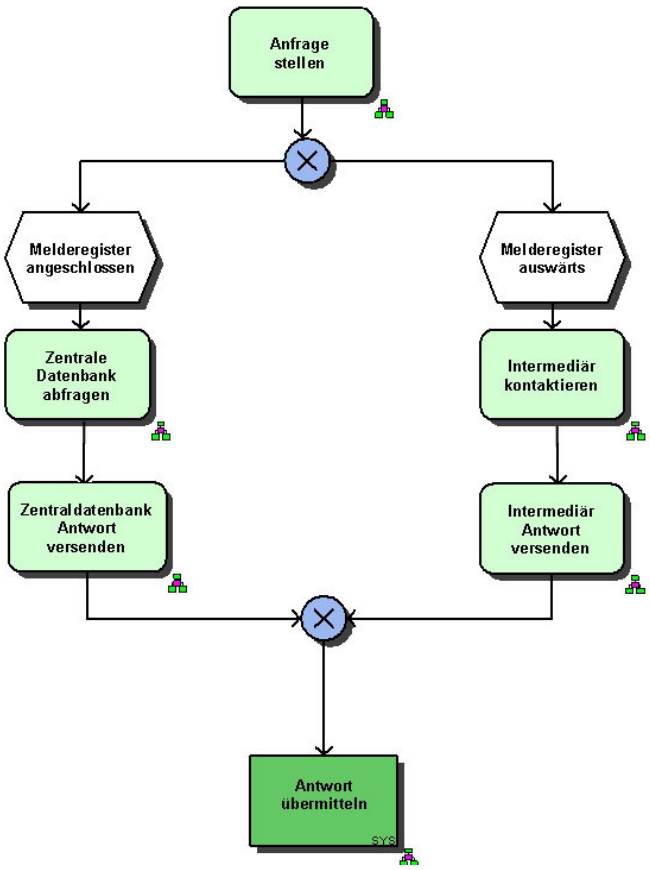


Abbildung 55: Das bereinigte fachliche Modell

Es ist leicht zu erkennen, dass die Komplexität des Modells sehr stark reduziert wurde. Dieses Modell, mit den den Funktionen hinterlegten eFZDs, bildet die Grundlage für eine Transformation in ein technisches Modell.

5.6.3 2. Schritt: Kontrollflussanalyse

Auf Basis des bereinigten Modells aus **Abbildung 55** wird nun die erste Funktion („Anfrage Stellen“) und das ihr hinterlegte eFZD analysiert. Der abgebildete Kontrollfluss lässt zwei semantisch äquivalente Umformungen zu. Einerseits eine Abbildung des Gesamtflusses anhand des <flow> oder des <sequence> Konstruktes. In diesem Fall wurde das <flow> Konstrukt gewählt. Wie im vorigen Kapitel in Abschnitt 4.3.3.2 beschrieben, wird der „split“ XOR Operator in ein <switch> Konstrukt umgewandelt. Die beiden Ereignisse „Melderegister angeschlossen“ und „Melderegister auswärts“ werden als Hinweise für die Entwickler als Kommentar und Bedingung in die korrespondierenden <case> „Äste“ eingetragen. Da der übrige Kontrollfluss innerhalb der beiden „Äste“ linear ist, und der XOR „join“ Operator wie in Abschnitt 4.3.3.3 beschrieben, gestrichen werden kann, müssen keine strukturierenden Aktivitäten mehr eingefügt werden. Das bis dahin entstandene BPEL Grundgerüst wird nun aufgezeigt.

```
<?xml version="1.0" encoding="UTF-8"?>
<bpws:process
  xmlns:bpws="http://schemas.xmlsoap.org/ws/2004/03/business-process/"
  expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
  name="eMRA" suppressJoinFailure="yes"
  targetNamespace="http://www.dvz-mv.de/eMRA">
  <bpws:partnerLinks>
  </bpws:partnerLinks>
  <bpws:variables>
  </bpws:variables>
  <bpws:correlationSets>
  </bpws:correlationSets>

  <bpws:flow name="eMRA">
    <bpws:links>
      <bpws:link name="Anfrage_stellenTOAnfrage_stellenXORSplit"/>
      <bpws:link name="XORJoinTOAntwort_uebermitteln"/>
    </bpws:links>
    <!-- Anfrage_stellen -->

    <bpws:switch name="Anfrage_stellenXOR">
      <bpws:targets>
        <bpws:target linkName="Anfrage_stellenTOAnfrage_stellenXORSplit"/>
      </bpws:targets>
      <bpws:sources>
```

```
        <bpws:source linkName="XORJoinTOAntwort_uebermitteln"/>
      </bpws:sources>
      <bpws:case>
        <bpws:condition><!--"MeldeRegister lokal"--></bpws:condition>
      </bpws:case>
      <bpws:otherwise>
      <!--"MeldeRegister auswärts"-->
      </bpws:otherwise>
    </bpws:switch>

    <!-- Antwort übermitteln-->

  </bpws:flow>
</bpws:process>
```

5.6.4 3. Schritt: Identifikation der relevanten Interaktionsschemata

Nun werden in der Reihenfolge des Auftauchens im Kontrollfluss die den Funktionen hinterlegten eFZDs analysiert. Das der ersten Funktion („Anfrage stellen“) hinterlegte eFZD ist in **Abbildung 44** abgebildet. Die einzige Schnittstelle zum Prozess bietet hierbei die „Prozessoperation“ „Antrag entgegennehmen“. Die ihr hinterlegten Attribute weisen sie als synchrone Operation aus. Daher muss ein <receive> Konstrukt erstellt werden, das sie abbildet.

```
<receive partnerLink="eGov_PortalPL" portType="eMRAPT"
         operation="Anfrage_entgegennehmen"
         variable="Anfrage_entgegennehmenInput" createInstance="yes"
         name="Antrag_entgegennehmen">
    <correlations>
        <correlation set="standard" initiate="yes"/>
    </correlations>
</receive>
```

Als nächstes zu betrachtendes Element erscheint im Kontrollfluss der XOR „split“ Operator. Wie in Abschnitt 4.3.3.2 des letzten Kapitels beschrieben, werden XOR „splits“ in <switch> Aktivitäten transformiert.

```
  <switch name="Anfrage_stellenXOR">
    <case condition="Melderegister_angeschlossen">
      activity
    </case>
    <otherwise>?
      <!-- Melderegister_auswaerts -->
      activity
```

```
    </otherwise>
  </switch>
```

Nun müssen die zwei zum Operator gehörenden Äste transformiert werden. In diesem Fall zuerst der linke. Die erste Funktion dieses linearen Kontrollflusses ist „Zentrale Datenbank abfragen“, das hinterlegte eFZD ist in **Abbildung 45** zu sehen. Dabei sendet der Prozess anhand der Operation „Anfrage weiterleiten“ eine Nachricht mit dem Namen „Anfrage“ vom Typ „0600“ an die Operation „Antrag weiterleiten“ der Zentralen Datenbank. Wie in Abschnitt 4.3.4 des letzten Kapitels beschrieben, wird das dargestellte Konstrukt in eine <invoke> Aktivität transformiert.

```
<invoke partnerLink="ZentraleDatenbankPL" portType="ZentraleDatenbankPT"
        operation="Antrag_weiterleiten" name="Anfrage_weiterleiten"
        inputVariable="Antrag_weiterleitenOutput">
</invoke>
```

Die nächste im Kontrollfluss vorkommende Funktion des linken Teilastes stellt die Funktion „Zentraldatenbank Antwort versenden“ dar. Das dazu äquivalente eFZD des rechten Kontrollflussastes ist in **Abbildung 48** zu sehen. Dabei schickt die Zentrale Datenbank anhand der Operation „Antwort versenden“ eine Nachricht mit dem Namen „Antwort“ vom Typ „0601“ an die „Antwort entgegennehmen“ Operation des Prozesses. Diese Art von eFZD wird wie in Abschnitt 4.3.4 des letzten Kapitels in eine <receive> Aktivität umgesetzt.

```
<receive partnerLink="ZentraleDatenbankPL" portType="ZentraleDatenbankPT"
         operation="Antrag_entgegennehmen" variable="Antrag_entgegennehmenInput"
         createInstance="no" name="Antwort_entgegennehmen">
    <correlations>
        <correlation set="standard2" initiate="yes"/>
    </correlations>
</receive>
```

Der XOR „join“ Operator ist das nächste im Kontrollfluss zu betrachtende Konstrukt. Da sein korrespondierender Vorgänger, der XOR „split“ wie oben beschrieben, in eine <switch> Aktivität umgesetzt wurde, kann dieser Operator gestrichen werden. Auf die Umsetzung des rechten Teilastes des Diagramms wird in dieser Beschreibung verzichtet, da dieser Ast bis auf die Namensgebung exakt wie der soeben beschriebene Part umgesetzt wird. Somit ist die letzte zu betrachtende Funktion des Kontrollflusses die „Antwort übermitteln“ Funktion. Das ihr hinterlegte eFZD ist in **Abbildung 49** zu sehen. Wie am Anfang der EPK wird auch in diesem Modell wieder eine „Prozessschnittstelle“ dargestellt. Da beide Operationen, über das neu eingeführte Attribut „synchron“, als synchrone Schnittstelle definiert wurden, muss die Operation als <reply> umgesetzt werden.

```
<reply partnerLink="eGov_PortalPL" portType="eMRAPT"
       operation="Antwort_uebermitteln"
       variable="Antwort_uebermittelnOutput"
       name="Antwort_uebermitteln">
    <correlations>
        <correlation set="standard" initiate="no"/>
    </correlations>
</reply>
```

5.7 Weitere Phasen

Da die folgenden Phasen nicht den Schwerpunkt dieser Arbeit darstellen, wird auf sie nur kurz eingegangen.

Integrationstest

Im vorliegenden Fall wurde ein Integartionstest nur rudimentär durchgeführt. Der Prozess als Ganzes wurde auf seine Funktionstüchtigkeit überprüft und die ausgetauschten Nachrichten analysiert. Nachdem der Prozess korrekt ablief, wurde auf weitere Tests verzichtet.

Betrieb

Da es sich bei dem erstellten Anwendungssystem um einen Prototyp handelt, wurden keine besonderen Vorkehrungen oder Schulungen angeboten.

Monitoring/Benchmarking

Auf das Monitoring und Benchmarking wurde in dieser Arbeit verzichtet.

Optimierung

Da keine Monitoring Daten gesammelt wurden, konnte bis zur Abgabe dieser Arbeit kein Optimierungspotential identifiziert werden.

6 Zusammenfassung - Fazit

Durch den Einfluss der Globalisierung und dem daraus resultierenden Kostendruck strukturieren sich viele Unternehmen zunehmend von Aufbauorganisierten- zu Prozess- / Ablauforientierten- Organisationen um. Da Geschäftsprozesse sehr volatile Modelle sind, sollte die zugrunde liegende, korrespondierende Softwarearchitektur ebenfalls sehr flexibel sein. Die Service-orientierte Architektur bietet diese Flexibilität. Änderungen, die am fachlichen Geschäftsprozessmodell durchgeführt werden, sollten so schnell wie möglich in ein technisches Modell des Geschäftsprozesses umgesetzt werden können. In diesem Zusammenhang wurde im Forschungsprojekt „Integration Engineering“ eine Methodik entwickelt, die dieser Arbeit zugrundegelegt wird.

Im Rahmen dieser Arbeit wurde, basierend auf dieser „Integration Engineering“ Methodik, ein Vorgehensmodell für die Integration von verteilten Anwendungssystemen erarbeitet. Um eine Redundanz der Modellierung zu verhindern, beinhaltet dieses Vorgehen einen Algorithmus, der die automatisierte Transformation eines fachlichen Geschäftsprozessmodells in ein technisches Modellgerüst ermöglicht, das durch die so genannte „Business Process Execution Language for Web Services“ (BPEL4WS) beschrieben wird. Unter Zuhilfenahme vorhandener Algorithmen können nun Geschäftsprozessmodelle, die anhand der „Integration Engineering“ Methodik erstellt wurden, auf ihre Transformierbarkeit validiert werden. Im Zuge dieser Transformation eines „Integration Engineering“ Geschäftsprozessmodells wurde die vorhandene „Integration Engineering“ Methodik analysiert. Bei der Analyse der Methodik wurden im Rahmen dieser Arbeit einige Schwachstellen und Uneindeutigkeiten identifiziert. So wurde in der „alten“ Methodik, der BPEL-Prozess nicht explizit dargestellt. Die Kommunikation mit dem Prozess konnte so nur angenommen werden. Deshalb wurden neue Symboltypen eingeführt, die den Prozess und die durch ihn zu Verfügung gestellten Operationen darstellen und diese Ambiguität beseitigten. Weiterhin konnte nicht zwischen synchroner und asynchroner Kommunikation zwischen den am Prozess beteiligten Partnern unterschieden werden. Zu diesem Zwecke wurden den vorhandenen Modellbausteinen zusätzliche Attribute hinzugefügt, die es nun ermöglichen diesen Sachverhalt darzustellen. Zusätzlich wurden Restriktionen an die Modellierung von Geschäftsprozessen formuliert, die eine Gewährleistung der Transformation sicher stellen sollen. Nach der Analyse und Erweiterung der Methodik, wurde die Methodik sowie die Transfomation anhand eines Praxisbeispiels aus dem E-Government Umfeld evaluiert. Das Beispielszenario wurde prototypistisch implementiert. Anhand dieses Beispielszenarios wurde sowohl die Transformation als auch die erweiterte „Integration Engineering“ Methodik und das Vorgehensmodell erfolgreich evaluiert.

Das Ziel der fachlichen Geschäftsprozessmodellierung und ihrer technischen Entsprechung sollte eine Art „round-trip“ Modellierung sein. In diesem Zusammenhang ist der momentane Stand der Forschungsarbeit in diesem Feld jedoch noch nicht fortgeschritten genug. Die „Retransformation“ von einem technischen Modell zurück zum fachlichen Modell unter Erhaltung der technischen Details ist bisher noch nicht ausreichend erforscht. Selbst in Feldern, wie beispielsweise der Werkzeugunterstützung der UML, sind konsistente Mo-

delle nicht immer gegeben. Um die Lücke zwischen dem fachlichen und dem technischen Modell zu schließen, wäre es wünschenswert ein Werkzeug auf einer Datenbasis, sowohl für eine fachliche als auch für eine technische Modellierung, heranziehen zu können. Dies könnte unter Umständen auch die Problematik lösen, die sich durch eine Änderung des fachlichen Geschäftsprozessmodells und der darauf folgenden Transformation ergibt. Die Erweiterungen des technischen Modells die zuvor durchgeführt wurden, gehen bei einer erneuten Transformation im Moment noch verloren.

7 Literaturverzeichnis

[ACD+03] Andrews, T. ; Curbera, F. ; Dholakia, H. ; Goland, Y. ; Klein, J. ; Leymann, F. ; Liu, K. ; Roller, D. ; Smith, D. ; Thatte, S. ; Trickovic, I. ; Weerawarana, S.: *Business Process Execution Language for Web Services version 1.1 (BPEL4WS)*. OASIS, May 2003. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[ACKM04] Alonso, G. ; Casati, F. ; Kuno, H.i ; Machiraju, V.: *Web Services, Concepts, Architectures and Applications*. Axel Springer Verlag, 2004

[AGB03] Atkinson, C. ; Gross, H-G ; Barbier, F.: Component Integration through built-in contract testing. In: Cechich, A. (Hrsg.): *Component-Based Software Quality*. Springer Verlag, 2003

[BDL+86] Bose, S. ; Duan, Z. ; Logvynovskiy, A. ; Shoniregun, C. ; Stirpe, P.A.: *Compilers: principles, techniques, and tools*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1986. – ISBN 0–201–10088–6

[Boe88] Boehm, Barry: A Spiral Model of Software Development and Enhancement. In: *IEEE Computer* Vol.21 (1988), Mai, Nr. 5, S. 61–72

[BPE] BPELIssues: *WS BPEL issues list*. Internet. http://www.choreology.com/external/WS_BPEL_issues_list.html. – Abgerufen am 21.08.2006

[Bun01] Bundesministerium der Justiz: *Gesetz über Rahmenbedingungen für elektronische Signaturen (SigG)*. Januar 2001. http://www.gesetze-im-internet.de/sigg_2001/index.html

[Bun03] Bundesministerium des Innern: *Bundesdatenschutzgesetz (BDSG)*. Januar 2003. http://www.datenschutz-berlin.de/recht/de/bdsg/bdsg03.htm

[Bun04] Bundesministerium des Innern: *Deutschland-Online - Die gemeinsame eGovernment-Strategie von Bund, Ländern und Kommunen*, August 2004. http://www.deutschland-online.de

[Bun05a] Bundesministerium der Justiz: *Melderechtsrahmengesetz (MRRG)*. September 2005. http://www.gesetze-im-internet.de/mrrg/index.html

[Bun05b] Bundesministerium des Innern: *E-Government-Initiative BundOnline 2005*. Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt), Jan 2005. http://www.kbst.bund.de/Content/Egov/Initiativen/Bol/bol.html

[Bun05c] Bundesministerium des Innern: *Standards und Architekturen für E-Government-Anwendungen*. Band 82. Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt), September 2005. http://www.kbst.bund.de/saga

[CCF+05a] Cabrera, L. F. ; Copeland, G. ; Feingold, M. ; Thatte, S. ; et., Al: *Web Services Atomic Transaction (WS-AtomicTransaction)*. OASIS, August 2005. http://specs.xmlsoap.org/ws/2004/10/wsat/

[CCF+05b] Cabrera, L. F. ; Copeland, G. ; Feingold, M. ; Thatte, S. ; et., Al: *Web Services Business Activity Framework (WS-BusinessActivity)*. OASIS, November 2005. http://schemas.xmlsoap.org/ws/2004/10/wsba/

[CCMW01] Christensen, E. ; Curbera, F. ; Meredith, G. ; Weerawarana, S.: *Web Services Description Language (WSDL) 1.1*. W3C, März 2001. http://www.w3.org/TR/wsdl

[CFFF05] Cabrera, L. F. ; Feingold, M. ; Freund, R. W. ; Freund, T.: *Web Services Coordination (WS-Coordination)*. OASIS, August 2005. http://specs.xmlsoap.org/ws/2004/10/wscoor/

[DG96] Dittrich, K. R. ; Gatziu, S.: *Aktive Datenbanksysteme: Konzepte und Mechanismen*. Bonn, 1996

[Din05] Dinkatt, Dipl.-Inf. M.: *Foliensatz: Blockseminar eGovernment, Fachhochschule Stralsund*. www.dvz-mv.de: DVZ Mecklemburg-Vorpommern, Oktober 2005

[DS05] Drawehn, J. ; Specht, T.: Kooperative Geschäftsprozesse: Modellierung und Ausführbarkeit. In: *Umsetzung von kooperativen Geschäftsprozessen auf eine internetbasierte IT-Struktur: Arbeiten aus dem Forschungsvorhaben Integration Engineering*. Leipzig, 2005, S. 97–105

[dvz] *Datenverarbeitungs Zentrale Mecklemburg Vorpommern (DVZ)*. http://www.dvz-mv.de

[FFG+04] Flender, C. ; Freytag, T. ; Gegenheimer, S. ; Landes, S. ; Nagy, A. ; Orts, S. ; Pohl, T.: *Workflow Petrinet Designer*. 0.8.0, 2004. http://www.woped.org

[FTH05] Fähnrich, K-P. ; Thränert, M. ; (Hrsg.), P. W.: *Leipziger Beiträge zur Informatik*. Bd. III: *Umsetzung von kooperativen Geschäftsprozessen auf eine internetbasierte IT-Struktur: Arbeiten aus dem Forschungsvorhaben Integration Engineering*. Leipzig, 2005

[FWT+04] Fallside, D. C. ; Walmsley, P. ; Thompson, H. S. ; Beech, D. ; Maloney, M. ; Mendelsohn, N. ; Biron, P. V. ; Malhotra, A. ; Holstege, M. ; Vedamuthu, A. S.: *XML Schema Version 1.0*. W3C, October 2004. http://www.w3.org/XML/Schema

[GHS95] Georgakopulus, D. ; Hornick, M. F. ; Seth, P.: *An overwiew of workflow management: From process modelling to workflow automation infrastructure*. Distributed and Parallel Databases. Apr 1995

[HC93] Hammer, M. ; Champy, J.: *Reengineering the Corporation*. harper Collins Pub. Inc., 1993

[HU72] Hecht, M. S. ; Ullman, J. D.: Flow graph reducibility. (1972), S. 238–250. http://dx.doi.org/10.1145/800152.804919. – DOI 10.1145/800152.804919

[KFG02] Krallmann, H. ; Frank, H. ; Gronau, N.: *Systemanalyse im Unternehmen. Vorgehensmodelle, Modellierungsverfahren und Gestaltungsoptionen*. R. Oldenbourg, Berlin, 2002. – Seite 344-346

[KK03] Kroll, P. ; Kruchten, P.: *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. 1st. Addison Wesley Professional, 2003 (The Addison-Wesley Object Technology Series)

[KNS92] Keller, G. ; Nüttgens, M. ; Scheer, A.-W.: *Semantische Prozeßmodellierung*. Veröffentlichungen des Instituts für Wirtschaftsinformatik Saarbrücken, Nr. 98, 1992

[Kop05] Kopp, O.: *Abbildung von EPKs nach BPEL anhand des Prozessmodellierungswerkzeugs Nautilus*. IAAS, Institut für Informatik, Universität Stuttgart, 2005

[Lan07] Landtag Mecklenburg-Vorpommern: *Meldegesetz für das Land Mecklenburg-Vorpommern (Landesmeldegesetz - LMG)*. Juli 2007. http://mv.juris.de/mv/MG_MV_rahmen.htm

[Ley01] Leymann, F.: *WSFL: Web Service Flow Language*. IBM http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[LK06] Lawrence, K. ; Kaler, C. ; Nadalin, Anthony (Hrsg.) ; Kaler, Chris (Hrsg.) ; Monzillo, Ronald (Hrsg.) ; Hallam-Baker, Phillip (Hrsg.): *Web Services Security: SOAP Message Security 1.1*. OASIS http://www.oasis-open.org/specs/index.php#wssv1.1

[LR99] Leymann, F. ; Roller, D.: *Production Work Flow: Concepts and Techniques*. Prentice Hall, 1999

[Mey88] Meyer, B.: *Object-Oriented Software Construction*. Prentice Hall, 1988

[Mit06] Mitschang, B.: *Skript: Grundlagen der Datenbanken und Informationssystemen*. Universität Stuttgart, IPVS Abteilung Anwendersoftware, WS 05/06

[MLM+06] MacKenzie, C. M. ; Laskey, K. ; McCabe, F. ; Brown, P. ; Metz, R.: *Reference Model for Service Oriented Architecture 1.0 Public Review Draft 1.0, 10 February 2006*. OASIS Commitee. – http://www.oasis-open.org/committees/download.php/16628/wd-soa-rm-pr1.pdf. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

[Mye01] Myers, G. J.: *Methodisches Testen von Programmen, The Art of Software Testing.* John Wiley & Sons, 2001

[OAS] OASIS: *Organization for the Advancement of Structured Information Standards.* OASIS Commitee, http://www.oasis-open.org

[OAS05] OASIS: *UDDI Version 3 Specification.* V 3.0.2. Feburary 2005. – http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htmRAUTEüddiv3

[OSCa] OSCI Leitstelle Bremen: *OSCI.* http://www.osci.de

[OSCb] OSCI Leitstelle Bremen: *OSCI-Transport.* Version 1.2. http://www1.osci.de/sixcms/detail.php?gsid=bremen02.c.1205.de

[Pet62] Petri, C. A.: *Kommunikation mit Automaten*, Institut für Instrumentelle Mathematik, Universität Bonn, Diss., 1962

[SGL96] Sreedhar, V. C. ; Gao, G. R. ; Lee, Y-F.: Identifying loops using DJ graphs. In: *ACM Trans. Program. Lang. Syst.* 18 (1996), Nr. 6, S. 649–658. http://dx.doi.org/10.1145/236114.236115. – DOI 10.1145/236114.236115. – ISSN 0164–0925

[SJW05] Scheer, A.-W. ; Jost, W. ; Wagner, K.: *Von Prozessmodellen zu lauffähigen Anwendungen.* VIII. Springer-Verlag, 2005 (ARIS in der Praxis)

[SNZ95] Scheer, A-W. ; Nüttgens, M. ; Zimmermann, V.: Rahmenkonzept für ein integriertes Geschäftsprozeßmanagement. In: *Wirtschaftsinformatik* 37 (1995), Nr. 5, S. 426–434

[SNZ98] Scheer, A-W. ; Nüttgens, M. ; Zimmermann, V.: *ARIS-Modellierungsmethoden, Metamodelle, Anwendungen.* Springer, 1998

[SNZ02] Scheer, A-W. ; Nüttgens, M. ; Zimmermann, V.: *Prozessmodellierung mit ARIS.* Bd. 1. Aufl. Vieweg Friedr. + Sohn, 2002. – S. 78

[SV05] Stahl, T. ; Völter, M.: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management.* dpunkt-Verl., 2005

[Tan95] Tannenbaum, A. S.: *Distributed Operating Systems.* International Editions. Prentice Hall, 1995. – ISBN:0131439340

[Tar73] Tarjan, R.: Testing flow graph reducibility. In: *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing.* New York, NY, USA : ACM Press, 1973, S. 96–107

[Tha01] Thatte, S.: *XLANG: Web Services for Business Process Design.* Microsoft Corp., 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[UM02] Unger, S. ; Mueller, .: Handling irreducible loops: optimized node splitting versus DJ-graphs. In: *ACM Trans. Program. Lang. Syst.* 24 (2002), Nr. 4, S. 299–333. http://dx.doi.org/10.1145/567097.567098. – DOI 10.1145/567097.567098. – ISSN 0164–0925

[W3C99] W3C: *XML Path Language (XPath) Version 1.0*. November 1999. http://www.w3.org/TR/xpath

[WCL+05] Weerawarana, S. ; Curbera, F. ; Leymann, F. ; Storey, T. ; Ferguson, D.: *Web Services Platform Architecture*. Prentice Hall, 2005

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfasst und nur die
angegebenen Quellen benutzt zu haben.

(Patrick M. Schneider)