Ein methodisches Framework zur Entwicklung offener, unternehmensweiter Systeme und Anwendungen mit Business Objects

Dissertation

Sascha Molterer
Institut für Informatik
Technische Universität München

Technische Universität München
Institut für Informatik

Ein methodisches Framework zur Entwicklung offener, unternehmensweiter Systeme und Anwendungen mit Business Objects

Sascha Molterer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Christoph Zenger

Prüfer der Dissertation: 1. Univ.-Prof. Dr. Manfred Broy
2. Univ.-Prof. Dr. Franz Lehner,
Universität Regensburg

Die Dissertation wurde am 12.03.2002 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 8.07.2002 angenommen.

Danksagung

Viele Menschen haben – direkt oder indirekt – dazu beigetragen, daß diese Arbeit entstanden ist. Mein Dank gebührt zuerst Prof. Dr. Manfred Broy, der mir am Lehrstuhl für Software- und Systems-Engineering im Rahmen des Forschungsverbundes FORSOFT die Arbeit an diesem interessanten und spannenden Thema ermöglichte, für das in mich gesetzte Vertrauen, für die großen Freiräume, die er mir bei der Verwirklichung meiner Ideen gewährte und für die Unterstützung und die Geduld bei der Durchführung der Arbeit. Herrn Prof. Dr. Franz Lehner danke ich für die Übernahme des Zweitgutachtens.

Ich möchte mich auch bei meinen Kollegen am Institut für Informatik und bei meinen langjährigen Projektpartnern bei BMW bedanken, die stets für ein angenehmes Arbeitsklima sorgten, mich bei dem Projekt unterstützten und denen ich interessante Impulse für meine Arbeit verdanke. Besonders zu erwähnen sind Barbara Paech, Christian Salzmann, Frank Marschall, Alexander Vilbig, Peter Braun und Günter Teubner am Institut und Dr. Albert Mas, Bernhard Huber, Markus Podolsky, Uwe Lammert, Stefan Million und Gerold Leeb bei BMW.

Weiterhin möchte ich mich bei allen Studenten bedanken, die im Rahmen ihrer Studienarbeiten sowie durch anregende Diskussionen und Implementierungsarbeiten zum Erfolg der vorliegenden Arbeit beigetragen haben. Hervorheben möchte ich an dieser Stelle Wiebe Hordijk und Markus Fröhler.

Besonderer Dank gilt meinen Eltern, die mich trotz chronischen Zeitmangels noch nicht ganz abgeschrieben und mir immer wieder viel Mut gemacht haben. Meinen größten Dank aber schulde ich Simone, die stets *da* war und immer an mich geglaubt hat.

Zusammenfassung

Sich ständig wandelnde Märkte, der Übergang von einem Anbieter- zu einem Kundenmarkt und die wachsenden Möglichkeiten der Informations- und Kommunikationstechnologien zwingen Unternehmen dazu, ihre Organisationsstrukturen, Arbeitsabläufe und Aufgabenverteilungen vollkommen neu zu ordnen. Statt einer hierarchischen, kontrollorientierten Aufteilung in spezialisierte Abteilungen, Funktionen und elementare Aufgaben richtet sich nun das Hauptaugenmerk der Unternehmen auf ihre unternehmensweiten und -übergreifenden Geschäftsprozesse. Darunter sind diejenigen Prozesse zu verstehen, die dem Kunden des Unternehmens eine Wertschöpfung bieten.

Die wesentlichen Vorteile für das Unternehmen sind flachere Hierarchiestrukturen, geringerer Kontrollaufwand und vor allem flexible, kundenorientierte Prozesse. Dieser Übergang ist aber nur dann möglich, wenn als Basis wandlungsfähige, prozeßorientierte Informationssysteme zur Verfügung stehen. Er ist nicht durchführbar auf Basis einer heterogenen Softwarelandschaft aus einer Vielzahl von unzureichend integrierten, monolithischen Informationssystemen. Vielmehr müssen die Abläufe, Aktivitäten, Aktoren und Informationen der Geschäftsprozesse des Unternehmens eindeutig auf unternehmensweit integrierte, offene Anwendungen abgebildet werden.

Die Arbeit leistet einen Beitrag zur methodischen Entwicklung offener, unternehmensweiter Informationssysteme. Erstmals werden dabei ein integriertes Metamodell, ein umfassender Satz an Beschreibungstechniken, ein unternehmensweites Vorgehensmodell und ein dreistufiges Werkzeugkonzept durchgängig auf Basis des Business Object Konzepts definiert. Vor allem die systematische Ableitung des in der Arbeit entwickelten Frameworks aus den Anforderungen des Business Process Reengineering, die nahtlose Integration arbeitsteiliger Vorgänge in Metamodell und Beschreibungstechniken sowie das neuartige Vorgehensmodell zur Entwicklung offener, unternehmensweiter Systeme auf Basis von Business Objects sind ein wichtiger Beitrag für die Forschung in diesem Bereich. Für den Praktiker liefert das entwickelte Framework eine Lösung, die auf akzeptierten Standards wie der MOF, der UML, dem Unified Process, der XML und der Enterprise Java Beans Infrastruktur basiert.

Inhaltsverzeichnis

Teil I

Einführung

Kapitel 1

Einleitung

Die Anforderungen an die Entwicklung von Informationssystemen in einem Unternehmen sind im letzten Jahrzehnt dramatisch gestiegen: Im Internetzeitalter wirken neue Kräfte auf Informationstechnologien und Systeme. Die stärkere Vernetzung im Unternehmen, zwischen den Unternehmen und mit Kunden und Zulieferern verlangt nach offenen, anpaßbaren Systemen. Als offene Informationssysteme werden in dieser Arbeit die Systeme bezeichnet, die aus Komponenten bestehen, deren Schnittstellen sowohl innerhalb des Informationssystems, als auch von anderen Systemen in gleicher Art und Weise zur Laufzeit genutzt werden können.

Immer neue Produkte, Geschäftsmöglichkeiten und Formen innerbetrieblicher Zusammenarbeit bedingen einen immer schnelleren Wandel der Anforderungen an die Funktionalität, Entwicklungszeit und den Wartungsaufwand von Informationssystemen. Heutige Entwicklungsmethoden und die daraus entstandenen Informationssysteme kommen diesen Anforderungen nur unzureichend nach: Zunächst werden große, wenig offene Systeme von Softwareherstellern gekauft und angepaßt. Diese Systeme sind dann Grundlage für die Abwicklung von Geschäftsprozessen in den verschiedenen Bereichen einer Unternehmung. Sollen Prozesse neuen Anforderungen angepaßt oder gar neue Prozesse ermöglicht werden, dann scheitert die Adaption dieser Systeme daran, daß die internen Abläufe nicht oder nur mit einem kleinen Freiheitsgrad geändert werden können.

Ist dieser Punkt erreicht, dann werden in eigenen Projekten individuelle Informationssysteme im Unternehmen erstellt, die teilweise auf den bestehenden Systemen aufsetzen, teilweise neue Geschäftsprozesse möglich machen, meistens aber schon vorhandene Funktionalität in anderer Form erneut als eigenes System ver-

wirklichen. Heutige Entwicklungsmethoden mit einer zu starken Konzentration auf das einzelne System führen dann auch dazu, daß in diesen Systemen eine eigene Sichtweise der Prozesse, eine speziell zugeschnittene Architektur und eine individuelle Festlegung der mehrmals verwendbaren Bausteine entwickelt wird.

Dies alles führt zu einer Softwarelandschaft, in der Geschäftsprozesse nicht durchgängig, kaum mehrfachverwendbar und redundant auf mehrere Systeme verteilt sind. Eine solche Softwarelandschaft kann mit dem Wandel der Anforderungen nur sehr schwer Schritt halten. Die fehlende Mehrfachverwendbarkeit bestehender Bausteine macht die Neuentwicklung von Systemen aufwendig und teuer. Aufgrund der redundanten, nicht durchgängigen Abbildung von Geschäftsprozessen auf mehrere Systeme bedeutet die Änderung eines Prozesses die Änderung mehrerer Systeme. Und schließlich bedeuten verschiedene Architekturen und die unterschiedliche Bindung der Funktionalität an Technik und Infrastruktur ein vermindertes Integrationspotential der einzelnen Systeme.

Um den Anforderungen gerecht zu werden, wird eine Umstellung der Softwarelandschaft in Unternehmen weg von heterogenen, geschlossenen Systemen hin zu offenen, unternehmensweiten Systemen notwendig. Offenen bedeutet hierbei, daß die fachlichen Dienste des Systems auch von anderen Systemen genutzt werden können. Unternehmensweit bedeutet, daß die fachlichen Dienste der Systeme von den unternehmensweiten Geschäftsprozessen des Unternehmens abgeleitet werden und nicht von lokalen Anforderungen zum Beispiel einer Abteilung.

Offene, unternehmensweite Systeme müssen deshalb in einem Prozeß entwickelt werden, der nicht nur das einzelne System, sondern auch die bestehenden, mehrfach verwendbaren Bausteine berücksichtigt. Mehrfach verwendbar heißt, daß die Bausteine schon von vornherein so entworfen wurden, daß ihre Funktionalität von mehreren Systemen genutzt werden kann. Die Abbildungen der Geschäftsprozesse in solchen Systemen und die Modellierung der Bausteine, die diese Geschäftsprozesse realisieren, muß unternehmensweit in einem einheitlichen Rahmen erfolgen. Offene, unternehmensweite Systeme müssen mit Werkzeugen modelliert, entwickelt und gewartet werden, die Technologie- und Infrastrukturunabhängig eine Neuentwicklung, Anpassung und Integration ermöglichen.

1.1 Ziel und Beitrag der Arbeit

Benötigt wird ein Framework als Rahmen, das ein an Mehrfachverwendung orientiertes Vorgehensmodell, eine gemeinsame Softwarearchitektur, eine einheitliche

Modellbildung und geeignete Werkzeuge zur Entwicklung offener, unternehmensweiter Systeme miteinander verbindet. Ansätze und Vorschläge für bestimmte Teile dieses Framework sind bereits in der Diskussion: Arbeiten wie [JGJ97], [AF98] oder [HS99] schlagen Methoden vor, mit denen Mehrfachverwendung systematischer in einen Entwicklungsprozeß einbezogen werden kann. Mehrstufige Architekturen [Gos98] und darauf aufsetzende Plattformen wie Enterprise JavaBeans [SUN00] ermöglichen aus infrastruktureller Sicht eine Aufteilung eines Systems in unternehmensweite und systemspezifische Aspekte. Workflowsysteme [JBS97] bieten die Möglichkeit, aus Geschäftsprozessen Workflows abzuleiten, zu modellieren und auszuführen. Business Objects [SPC+97, PSM98, PSM99] geben einen einheitlichen Rahmen vor, wie die fachlichen Dienste einer Unternehmung mehrfach verwendbar partitioniert und implementiert werden können.

Doch es reicht nicht aus, die einzelnen Ansätze ohne vorherige Anpassung und Erweiterung zusammenzustellen: Die genannten Methoden stellen zwar die Mehrfachverwendung in den Vordergrund, es fehlt aber eine stärkere Einbindung einer unternehmensweiten Architektur und die damit verbundenen Rollen und Aufgaben im Entwicklungsprozeß. Heutige Workflowsysteme unterstützen nur die Abbildung von Geschäftsprozessen auf die eigene Architektur, was zum einen eine mehrfache Verwendung der Workflows verhindert, zum anderen zu einem Bruch zwischen den Teilen der Geschäftsprozesse, die in Workflowsystemen realisiert sind, und den Teilen, die in den eigentlichen Informationssystemen realisiert sind, führt.

Die derzeitigen Business Objects Ansätze versuchen, die Abbildung der Geschäftsprozesse auf Informationssysteme zu vereinheitlichen. Allerdings sind die unterschiedlichen Definitionen, was Business Objects sind und wie man mit ihnen modelliert, entwirft und implementiert, nicht einheitlich und oftmals auf die Infrastruktursicht beschränkt. Auch gibt es unterschiedliche Modelle und Architekturen für Business Objects, je nachdem, ob diese Objekte zur Laufzeit oder zur Konstruktion verwendet werden sollen. Im letzteren Fall wird auch der Begriff „Business Component“ [HS99] verwendet. Und mit den aktuellen Werkzeugen können die Elemente eines offenen, unternehmensweiten Systems nur zum Teil und wenig zusammenhängend entwickelt und angepaßt werden.

Ziel dieser Arbeit ist deshalb ein integriertes Framework, mit dem offene, unternehmensweite Systeme systematisch, durchgängig und einheitlich entwickelt werden können. Ausgangspunkt und Basis ist dabei das Konzept des Business Objects.

Grundgedanke bei diesem Konzept ist es, jedes Element der Geschäftswelt, wie „Auftrag“, „Kunde“ oder „Vertrag“ in Analyse, Design und Implementierung für

sich auf ein Element der Systemwelt, dem Business Object, isomorph und einmalig abzubilden. Zusammen mit einer mehrstufigen Architektur ist mit diesem Konzept nicht nur eine einheitliche softwaretechnische Grundlage, sondern auch die Durchsetzung einer unternehmensweiten Richtlinie zur Partitionierung der Geschäftsprozesse gegeben. In dieser Arbeit wird ein Metamodell für Business Objects entwickelt, das als Basis für die Bestandteile des Frameworks dient.

Um solche Business Objects zu entwickeln und zu verwenden, sind Erweiterungen traditioneller, auf Einzelsysteme zentrierter Vorgehensmodelle notwendig. In dieser Arbeit werden neue Rollen und Organisationsformen für Projekte vorgestellt, die offene, unternehmensweite Anwendungen auf der Basis der Business Object Architektur entwickeln.

Da Workflows wie „Kundenreklamation bearbeiten“ auch Elemente der Geschäftswelt sind, müssen sie, damit kein Bruch zwischen Workflowsystem und Business Object System entsteht, auf Business Objects abgebildet werden. Die Arbeit gibt hierzu ein durchgängiges Modell an, mit dem Workflows als Business Objects modelliert und realisiert werden können.

Die Mehrfachverwendung von Business Objects ist auf mehrere Arten möglich: Entweder unternehmensweit innerhalb einer Unternehmung zur Laufzeit, indem ein Business Object aus der Sicht verschiedener Geschäftsprozesse von mehreren Systemen genutzt wird, oder unternehmensübergreifend, indem ein Business Object zur Entwicklungszeit in Systeme verschiedener Unternehmen übernommen wird, wie zum Beispiel das Business Object „Auftrag“ für die jeweiligen Auftragsverwaltungssysteme. Letzteres Vorgehen erfordert eine genaue und vor allem normalisierbare Spezifikation des Verhaltens eines Business Objects. Denn obwohl die meisten Geschäftsprozesse in Unternehmen im Groben den gleichen Ablauf haben, so werden sie im Detail jedoch von Unternehmen zu Unternehmen anders ausgeführt und durch Informationssysteme unterstützt. Deshalb muß vor der Nutzung eines Business Objects geklärt werden, ob das Verhalten des Business Objects auch den eigenen Abläufen entspricht und alle Anforderungen der schon vorhandenen Business Objects erfüllt werden. Eine rein syntaktische Prüfung über den Vergleich der Namen, Parameter und Typ der Schnittstellen genügt nicht. Der Vergleich zweier Semantik-Spezifikationen – einerseits die Spezifikation, was vom Business Object erwartet wird und andererseits die Spezifikation dessen, welches Verhalten das Business Object aufweist – ist zur Zeit weder in der Forschung umfassend geklärt noch praktikabel gelöst. Eine umfassende Lösung dieses Problems würde jedoch den Rahmen der Arbeit sprengen. Deshalb beschränkt sich diese Arbeit auf den unternehmensweiten Aspekt der mehrfachen Nutzung von Business Objects.

Um die Vorteile des Metamodells und seiner Erweiterungen für die Integration von Workflow und die Berücksichtigung verschiedener Arten der Wiederverwendung zu nutzen, ist ein geeignetes Werkzeugkonzept notwendig. Der Hauptfokus des Werkzeugkonzeptes, das in dieser Arbeit entwickelt wird, ist die Möglichkeit der Entwicklung von unternehmensweiten Anwendungen auf der Basis des Business Object Modells unabhängig von der technologischen Bewandtnis des Entwicklers. Das Werkzeug soll auch Fachexperten ermöglichen, bestimmte Aspekte des Systems, wie zum Beispiel die Business Regeln oder einen Workflow, zu ändern. Beitrag der Arbeit ist eine Abbildung des Metamodells auf eine geeignete Infrastruktur und darauf aufbauend ein Konzept für ein modellbasiertes Werkzeug zur Anpassung der offenen, unternehmensweiten Systeme.

Die Arbeit leistet einen Beitrag zur methodischen Entwicklung offener, unternehmensweiter Informationssysteme. Erstmals werden dabei ein integriertes Metamodell, ein umfassender Satz an Beschreibungstechniken, ein unternehmensweites Vorgehensmodell und ein dreistufiges Werkzeugkonzept durchgängig auf Basis des Business Object Konzepts definiert. Vor allem die systematische Ableitung des in der Arbeit entwickelten Frameworks aus den Anforderungen des Business Process Reengineering, die nahtlose Integration arbeitsteiliger Vorgänge in Metamodell und Beschreibungstechniken sowie das neuartige Vorgehensmodell zur Entwicklung offener, unternehmensweiter Systeme auf Basis von Business Objects sind ein wichtiger Beitrag für die Forschung in diesem Bereich. Für den Praktiker liefert das entwickelte Framework eine Lösung, die auf akzeptierten Standards wie der MOF, der UML, dem Unified Process, der XML und der Enterprise Java Beans Infrastruktur basiert.

1.2 Stand und Defizite der verwandten Arbeiten

Es wurde bereits erwähnt, daß einzelne Bausteine für das Framework bereits existieren: Arbeiten zu den Business Objects, Konzepte für eingebettete Workflowsysteme, Infrastrukturen und Werkzeuge und wiederverwendungszentrierte Methoden. Im folgenden werden die Arbeiten zu diesen Bausteinen dargestellt und diskutiert, wie sie in ein integriertes Framework passen und welche Erweiterungen notwendig sind.

1.2.1 Business Objects

Das Konzept des Business Objects gibt es, mit nahezu ähnlichen Definitionen, seit Einführung objektorientierter Methoden. In frühen Methoden [SM88] wurde der Begriff „Objekt“ für „Business Object“ verwendet: „An Object is an abstraction of a set of real-world things“ [SM88]. Allerdings hatten die frühen Methoden ihren Hauptfokus in der objektorientierten Analyse. In den späteren Phasen wurde kaum geachtet auf eine klare Abgrenzung von Objekten und den systemspezifischen Elementen. Die Eigenschaft von Business Objects, ein Element der Geschäftswelt isomorph in ein Element der Systemwelt in der Analyse, dem Entwurf und der Implementierung abzubilden, wurde dann ab 1994 in verschiedenen Büchern thematisiert, die solche Objekte dann explizit „Business Object“ oder „Business Component“ nannten. Zu nennen sind hier [SLJR94, Tay94, Sim94]. Allerdings enthalten diese Arbeiten noch kein fundiertes Modell für Business Objects und kein Konzept, wie die Infrastruktur für solche Objekte aufgebaut sein muß.

Fundierter wurde in der Object Management Group (OMG) Business Object Domain Task Force (BODTF) [OMG95] und in einem seit 1995 jährlich in Zusammenhang mit der OOPSLA stattfindenden Workshop „Business Object Design and Implementation“ [SPC+97, PSM98, PSM99] nach Definitionen und Architekturen für Business Objects gesucht. Bemerkenswert ist hier die Business Object Component Architecture (BOCA) [OMG98a]. In der BOCA wird ein Metamodell für Business Objects, das mit der Meta Object Facility (MOF) [OMG98d] definiert wurde, vorgestellt. Es sollte Grundlage für eine Standardisierung einer Business Object Facility (BOF) sein, einer Infrastruktur für Business Objects, die auf CORBA aufsetzt. Der Vorschlag wurde aber inzwischen abgelehnt, da zunächst der neue Standard der OMG für Server-seitige Komponenten „CORBA-components“ [OMG98b] abgewartet werden sollte.

Trotz der Ablehnung ist der BOCA-Vorschlag ein Ausgangspunkt für diese Arbeit. Da er aber vor allem zum Ziel hat, einen Standard für Domainarchitekturen auf der Basis von Business Objects zu sein, sind die damit zusammenhängenden Aspekte zu detailliert, andere, wie zum Beispiel die Wiederverwendungsaspekte zur Laufzeit, überhaupt nicht dargestellt. Darüberhinaus vermischt BOCA das Metamodell für Business Objects mit dem Modell der darunter liegenden Infrastruktur. Diese Abbildung sollte aber separat gehandhabt werden, um das Modell möglichst unabhängig der verwendeten BOF zu halten.

In jüngster Zeit, auch mit dem Erscheinen produktiv einsetzbarer Infrastrukturen wie den auf Enterprise JavaBeans [SUN00] basierenden Applikationsservern, er-

schienen auch Arbeiten, die sich mit dem Design und der Implementierung von Business Objects auseinandersetzen [ES98, Pri96, HS99]. Diesen Arbeiten fehlt aber ein fundiertes Modell, so daß sich ihre Ergebnisse nicht ohne weiteres übertragen lassen.

1.2.2 Methoden zur Entwicklung offener, unternehmensweiter Systeme

Es gibt nur wenige Methoden zur Entwicklung von Softwaresystemen, die eine unternehmensweite Sichtweise auf den Entwicklungsprozeß propagieren und dabei eine offene, auf Business Objects basierende Architektur zum Ziel haben. Zu nennen wären hier die Methode SELECT [AF98], die Weiterentwicklung der Methode OOSE [JGJ97], die kürzlich erschienene Business Component Factory Methode [HS99] sowie die Methode BOOSTER [Kor01]. In diesen Arbeiten werden aber die neuen Rollen und Organisationsformen für die Entwicklung offener, unternehmensweiter Systeme kaum oder gar nicht genannt. Die Ausrichtung der Methoden konzentriert sich überwiegend auf technische Aspekte und das Vorgehen im Kleinen.

Darüberhinaus konzentrieren sich die genannten Methoden auch fast ausschließlich auf die Sicht der Software-Industrie, nämlich auf eine unternehmensübergreifende Mehrfachverwendung von Business Object „Komponenten“ zur Konstruktion von Anwendungen verschiedener Unternehmen. Die Fragestellungen, die eine Mehrfachnutzung von Business Objects unternehmensweit für *ein* Unternehmen mit sich bringen, werden vernachlässigt. Außerdem wird in keinem der genannten Ansätze eine nahtlose Integration von arbeitsteiligen Vorgängen beziehungsweise Workflows in die auf Business Objects basierende Methode erreicht.

1.2.3 Workflowsysteme

Workflow-Management ist ein Thema im Software-Engineering mit vielen Verbindungen zu anderen Gebieten wie Prozeßmodellierung, Business Process Reengineering, CSCW oder Wirtschaftswissenschaften. Entsprechend umfangreich und fundiert sind die Arbeiten zu dem Thema.

Für diese Arbeit, vor allem für die Abbildung von Geschäftsprozessen auf Informationssysteme sind viele Forschungsergebnisse relevant, allerdings aus einem bestimmten Blickwinkel: Wie können Workflows in einem offenen, unterneh-

mensweiten System auf Basis von Business Objects unterstützt werden. Das heißt, wie können einerseits Workflowmanagement-Funktionen in das Business Object System eingebettet werden ohne neben dem eigentlichen Informationssystem ein eigenständiges, geschlossenes Workflowsystem zu betreiben zu müssen und andererseits, wie werden mit Business Objects die arbeitsteilige Vorgänge der Geschäftswelt abgebildet. Hierzu gibt es noch keine Arbeiten oder Lösungen. Verwandt sind Standards, Forschungsprototypen wie auch Industrieprodukte, in denen ein komponentenorientiertes Vorgehen bei der Entwicklung von Workflows-Management-Systemen realisiert wurde.

Standards, die ein komponentenorientiertes Workflowsystem zum Ziel haben, sind die Vorschläge der Workflow Management Coalition (WfMC) [WFM94] sowie der OMG jointFlow [OMG98c]. Als Forschungsprototypen sind zu nennen METEOR2 [KS97], MENTOR [WWWKD97], WorCOS [Sch99] oder WASA [Wes99]. Relevante Industrieprodukte sind iFlow [Fuj00] der Firma Fujitsu und Verve [Ver00] der gleichnamigen Firma.

Auf die Details muß hierbei nicht eingegangen werden, denn alle oben genannten Ansätze haben ein Merkmal gemeinsam: Trotz ihrer komponentenorientierten Struktur ist die Logik, die den Workflow definiert und die vom Workflow Enactment Service interpretiert wird, um den Workflow auszuführen, weiterhin ein Teil des Workflowsystems. Dies bedeutet: Auch wenn man ein solches System in eine Business Object Architektur einbetten würde, gäbe es einen Bruch zwischen der Logik, die in den Business Objects selbst realisiert ist, und der Logik, die vom Workflowsystem verwaltet wird. In dieser Arbeit wird ein eigenständige, neuartige Lösung vorgestellt, bei der die Business Objects selbst den Workflow repräsentieren und ausführen. Diese Lösung ist zudem eine Antwort auf die bisher unzureichend geklärte Frage, wie Workflow Definitionen und Instanzen von verschiedenen Systemen zur Laufzeit mehrfach genutzt werden können.

1.2.4 Infrastrukturen und Werkzeugunterstützung

Ein Teil der Arbeit beschäftigt sich mit einem geeigneten Werkzeugkonzept mit dem Schwerpunkt einer modellbasierten Anpassung bestehender unternehmensweiter Systeme. Verwandte Arbeiten zu diesem Thema gibt es noch nicht. Dazu muß auch eine Abbildung des entwickelten Metamodells auf die Infrastruktur angegeben werden.

Für eine Basis für Business Object Systeme kommen zur Zeit drei produktive Systemtypen in Frage: Applikationsserver auf Basis der Enterprise JavaBeans

Spezifikation [SUN00], der IBM ComponentBroker mit einem eigenen Komponentenframework, das Managed Object Framework (MOFw) oder ein Applikationsserver auf Basis der CORBAcomponents Spezifikation der OMG [OMG98b]. Da es noch keine Implementierung für letztere Spezifikation gibt und das MOFw nicht weit verbreitet ist, wird in dieser Arbeit das Enterprise JavaBeans Framework als Zielinfrastruktur verwendet.

Da modellbasierte Änderung bedeutet, daß der Benutzer nicht mehr den Quellcode, sondern das (unter Umständen visuelle) Modell seiner Anwendung anpaßt, wird noch eine geeignete Lösungsmöglichkeit zu einer komfortablen Umsetzung gesucht. Eine Möglichkeit ist die Erweiterung eines geeigneten CASE-Tools. Die Idee wäre dann, ein UML-Profil für das Business Object Metamodell zu erstellen und dieses dann als Modell für den Benutzer zu präsentieren.

1.3 Ergebnisse und Aufbau der Arbeit

Die wichtigsten Ergebnisse dieser Arbeit sind:

1. Es werden die Anforderungen des Business Process Reengineering auf den Entwurf und Entwicklung von Informationssystemen erarbeitet und gezeigt, daß die bisherigen, von Softwareunternehmen geprägten, projekt- und anwendungsfallbezogenen Methoden unzureichend sind für ein erfolgreiches Business Process Reengineering.

2. Offene, unternehmensweite Systeme werden als geeignetes Konzept vorgestellt, um unternehmensweite Geschäftsprozesse auf Informationssysteme abbilden zu können. Zur Entwicklung solcher Systeme wird ein Framework bestehend aus Metamodell, Beschreibungstechniken, Vorgehensmodell und Werkzeugkonzept vorgeschlagen.

3. Auf Grundlage des in dieser Arbeit eingeführten Begriffes der Mehrfachnutzung wird das Konzept der Business Objects als grundlegendes Element für die Modellierung und Realisierung von offenen, unternehmensweiten Systemen propagiert. Ein umfassendes Metamodell definiert und verbindet die einzelnen Aspekte eines Business Object Modells.

4. Erstmals wird ein Metamodell für Business Objects angegeben, das eine integrierte Modellierung von arbeitsteiligen Vorgängen ermöglicht und somit eine durchgängige und einheitliche Modellierung aller Elemente der Geschäftswelt ohne Brüche ermöglicht.

5. Auf Basis des Metamodells wird ein integrierter Satz an Diagrammtypen als Profil der UML angegeben, der verschiedene Sichten auf alle Aspekte eines Business Object Modells ermöglicht.

6. Ein eigenständiges Vorgehensmodell basierend auf dem Unified Process beschreibt Rollen, Aktivitäten und Phasen eines Prozesses zur Entwicklung offener, unternehmensweiter Systeme auf Grundlage des Business Object Metamodells und der entwickelten Modellierungstechniken. Es berücksichtigt dabei speziell das von anderen Modellen nicht abgedeckte Anwendungsfall-übergreifende und Geschäftsprozeß-orientierte Vorgehen zur Entwicklung von Informationssystemen für *ein* Unternehmen.

7. Basierend auf allen oben genannten Ergebnissen wird ein dreistufiges Werkzeugkonzept zur Modellierung eines Business Object Modells, zur Anreicherung des Modells um infrastrukturelle Komponenten und zur Abbildung des Modells auf eine Business Object Infrastruktur vorgestellt. Es umfaßt neben der XML-basierten eXtended Business Object Definition Language (XBODL) eine Abbildung aller Metamodellkonzepte auf die Enterprise Java Beans Infrastruktur.

Im folgenden Kapitel wird auf die Anforderung des Business Reengineering an die Entwicklung unternehmensweiter Anwendung eingegangen und dargestellt, daß derzeitige Methoden und Ansätze diese Anforderung nicht oder nur teilweise erfüllen. Anschließend wird das Konzept zu dem in dieser Arbeit vorgestellten Framework diskutiert.

Das Framework, wie in Abbildung 1.1 dargestellt, basiert hauptsächlich auf dem Business Object Konzept. Grundideen und bestehende Ansätze werden in Kapitel 3 vorgestellt und diskutiert. Es werden Kriterien aufgestellt, die für ein geeignetes Business Object Modell erfüllt sein müssen. Ein zweiter Grundgedanke des Frameworks ist die Integration von Workflow- und Anwendungsmodellierung. Die Anforderungen an eine Realisierung von Workflow-Funktionalität in Systemen und Ansätze in diesem Bereich werden im Kapitel 4 dargestellt und anhand der Kriterien bewertet. Zusammen mit den in Kapitel 5 getroffenen Entwurfsgrundsätzen bilden die Kriterien und die Anforderungen den Rahmen für den Aufbau und die Struktur des Business Object Metamodells.

Anschließend wird das eigentliche Framework vorgestellt. Das Framework gliedert sich in vier Teile: Zum ersten wird das Business Object Metamodell als Grundlage aller anderen Elemente des Frameworks in Kapitel 6 entwickelt. Das Metamodell ist die Basis und der Integrationspunkt des gesamten Frameworks. Zum zweiten wird zur Beschreibung der Modelle des Metamodells in Kapitel 7

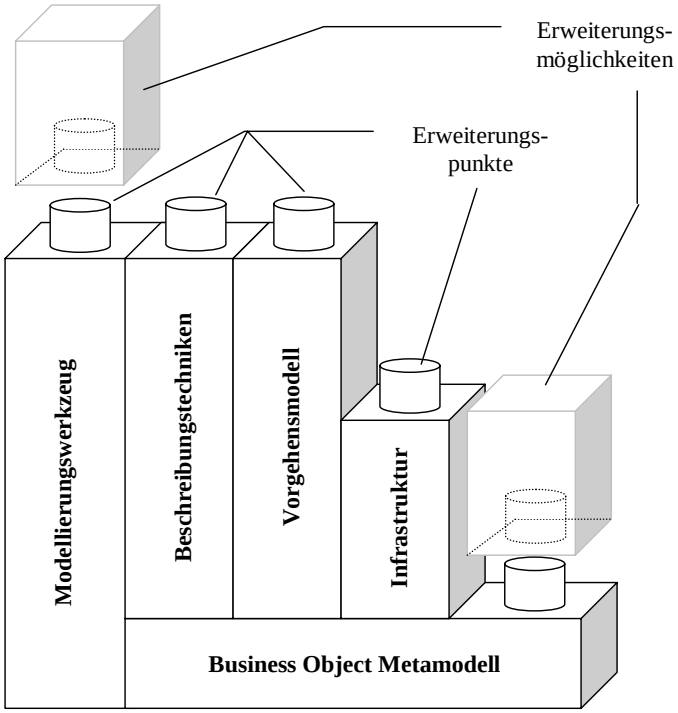

Abbildung 1.1: Das Framework in der Übersicht

ein Profil der UML definiert. Das Profil umfaßt mehrere Sichten und deckt alle Aspekte der Ausdrucksfähigkeit des Metamodells ab. Zum dritten wird ein Vorgehensmodell basierend auf diesem Profil in Kapitel 8 vorgestellt. Es basiert auf dem Unified Process, bringt die Beschreibungstechniken des Profils in einen Zusammenhang und definiert den Prozeß zur Entwicklung unternehmensweiter Systeme und Anwendungen.

In Kapitel 9 wird schließlich ein integriertes Werkzeugkonzept vorgestellt. Es umfaßt neben dem Modellierungswerkzeug – einer Erweiterung des kommerziellen Werkzeugs Together – ein Tool zur Abbildung des Modells auf eine Infrastruktur. In dieser Arbeit wurden Enterprise JavaBeans als Infrastruktur gewählt. Abschließend werden in Kapitel 10 die Ergebnisse der Arbeit zusammengefaßt und diskutiert sowie Erweiterungsmöglichkeiten aufgezeigt.

Teil II

Grundlagen

Kapitel 2

Entwicklung unternehmensweiter Systeme

Business Process Reengineering ist ein Ansatz zur Geschäftsprozeß-orientierten Gestaltung von Unternehmen. Ziel ist eine fundamentale Neuausrichtung der Unternehmensstrukturen auf Kunden-orientierte Geschäftsprozesse und die Auflösung der traditionellen organisatorischen Grenzen hin zu einer Fokussierung auf die Kerngeschäftsprozesse. In diesem Kapitel wird motiviert, warum ein Business Reengineering auch neue Anforderungen an den Aufbau und die Entwicklung von Informationssystemen stellt, die durch konventionelle Architekturen und Methoden nicht erfüllt werden. Es wird deshalb in dieser Arbeit ein methodisches Framework zur Entwicklung von offenen, unternehmensweiten Anwendungen auf der Basis von Business Objects als Grundlage eines Business Process Reengineering entwickelt. Dieses wird am Schluß des Kapitels in der Übersicht vorgestellt.

2.1 Business Reengineering

In den letzten Jahren hat sich die Bedeutung der Rolle der Organisation eines Unternehmens dramatisch gewandelt. War bisher die Hauptaufgabe von Organisationsstrukturen, Mitarbeitern bestimmte Funktionen und Aufgaben innerhalb einer Aufgabenkette zuzuordnen, wie es in Abbildung 2.1 dargestellt wird, treten seit den 1980er Jahren – ausgehend von einem Bedürfnis nach einer effektiven und effizienten Nutzung der Unternehmensressourcen – die wesentlichsten Geschäftstätigkeiten in den Vordergrund aller Betrachtungen. Porters' Modell der Wertschöpfungsketten [Por98] trug dem als erstes Rechnung: Es teilte die Ge-

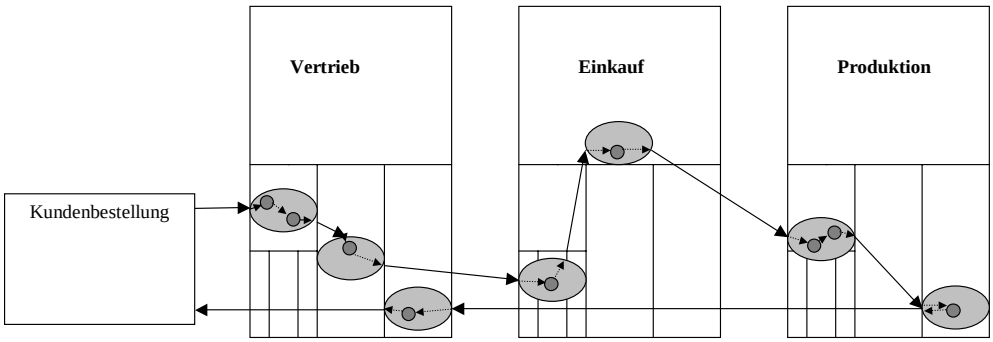

Abbildung 2.1: Prozeßfragmentierung

schäftstätigkeiten in wertschöpfende, primäre Aktivitäten und unterstützende, sekundäre Aktivitäten ein. Um die Wettbewerbsfähigkeit eines Unternehmens zu erhöhen, sollten alle sekundären Aktivitäten vermieden werden. Dies führte dann dazu, daß sekundäre Aktivitäten von Unternehmen „outgesourced“ wurden.

Seit den Anfängen der 1990er Jahren wurden auf der Basis dieses Modells verschiedene Reorganisierungsansätze entwickelt, die unter dem Begriff „Business Process Reengineering“ (BPR) bekannt wurden. Die bekannteste und prägendste Publikation zu diesem Thema ist das gleichnamige Buch von Hammer und Champy [HC93]. Business Process Reengineering steht für eine fundamentale Neuausrichtung der Unternehmensstrukturen auf Kunden-orientierte Geschäftsprozesse. Die traditionellen organisatorischen Grenzen wurden zugunsten einer Fokussierung auf diese Kerngeschäftsprozesse aufgelöst.

2.1.1 Unternehmensprozesse als Ausgangspunkt

Definition 2.1 (Geschäftsprozeß) *Ein Geschäftsprozeß ist eine logisch zusammenhängende Menge von Teilschritten, die auf das Erreichen eines wertschöpfenden Ergebnisses eines Unternehmens ausgerichtet ist. Ausgelöst durch ein Ereignis werden (materielle und/oder immaterielle) Produkte unter Betrachtung von Vorgangs- und Ausführungsregeln transformiert oder erzeugt. Ein Geschäftsprozeß ist dabei unabhängig von Abteilungs- oder Funktionsgrenzen.*

Unternehmensprozesse werden in [HC93] als Bündel von Aktivitäten, für das ein oder mehrere unterschiedliche Inputs benötigt werden und das für den Kunden ein Ergebnis von Wert erzeugt, definiert. Sie sind der Ausgangspunkt für die Veränderung von großen Teilbereichen und verschiedenen Aspekten einer Organisation.

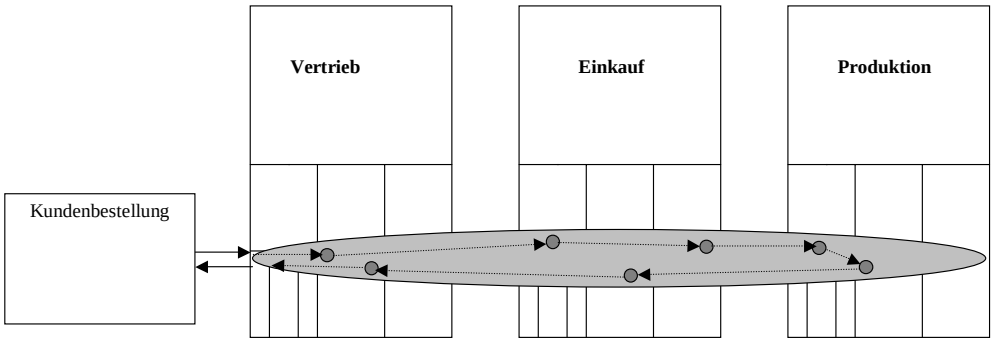

Abbildung 2.2: Unternehmensweite Geschäftsprozesse

Zu den Veränderungen gehören nach [HC93]:

- Aus eng definierten, aufgabenorientierten Positionen werden multidimensionale Zuständigkeiten für Prozesse. Damit ändern sich organisatorische Einheiten; Fachabteilungen werden Prozeßteams.
- „Empowerment“ der Mitarbeiter ist das neue Motto, daß heißt: Mitarbeiter sollen selber Regeln definieren können, statt nur Regeln zu befolgen. Auf diese Weise sollen zu aufwendige Kontrollhierarchien vermieden werden.
- Hierarchische Organisationsstrukturen weichen flachen, weil durch die nicht mehr fragmentierten unternehmensweiten Prozesse keine Stellen mehr benötigt werden, die als Bindeglied zwischen den Fragmenten fungieren.

Nach einem Business Process Reengineering spielen funktionale und organisatorische Einheiten nur noch eine untergeordnete Rolle, wie in Abbildung 2.2 dargestellt. Das Entscheidende ist die Konzentration auf die Geschäftsprozesse eines Unternehmens.

2.1.2 Die Rolle der Informationstechnologie

Diese Veränderungen und die Ausrichtung der Unternehmung nach den Geschäftsprozessen zieht bestimmte Anforderungen an die unterstützenden Informationssysteme (Prozeß, Architektur, Abbildung) nach sich.

Definition 2.2 (Informationssystem (rechnergestütztes)) *Ein (rechnergestütztes) Informationssystem ist ein Teilsystem aller Prozesse der Gewinnung, Speicherung, Verarbeitung und Übermittlung von Informationen und der an diesen*

Prozessen beteiligten Menschen und Maschinen in ihrer informationsverarbeitenden Rolle. Dabei ist die Erfassung, Speicherung, Übertragung, Darstellung und/oder Transformation von Information durch den Einsatz von Rechenanlagen automatisiert.

Mit der verstärkten informations- und kommunikationstechnischen Vernetzung in der betrieblichen Praxis wächst die Bedeutung des Produktionsfaktors Information. Insbesondere mit der Ausrichtung auf unternehmensweite Prozesse sind geeignete Informationssysteme zu deren Unterstützung unabdingbar [Leh99]. Hammer und Champy weisen der Informationstechnologie beim Business Process Reengineering eine tragende Rolle zu.

Allerdings wird nicht sofort jeder Geschäftsprozeß durch die bloße Einführung und Nutzung der Informationstechnologie bestmöglich unterstützt. Im Gegenteil: In [PRW96] wird das sogenannte Produktivitätsparadoxon beschrieben, das keinen Zusammenhang zwischen den Investitionen in die Informationstechnik und der Produktivität feststellt. Demnach fehlt eine Korrelation zwischen Produktivität und steigender Investition in die Informationstechnik.

Verschiedene Erklärungsansätze für dieses Phänomen sind in der Diskussion. Gründe, die die Informationstechnologie an sich betreffen, sind nach [PRW96]:

- eine technikzentrierte Sichtweise bei der Planung und Realisierung von Informationssystemen,
- eine bloße „Elektrifizierung“ bestehender Abläufe ohne Anpassung der Prozeßstruktur, deren Folge Ineffizienzen und eine mögliche Zementierung der Organisationsstrukturen sind, sowie
- eine Behinderung neuer Prozesse durch bestehende Informationssysteme, die die alte fragmentierte Prozeßstruktur widerspiegeln. Dies bedeutet umgekehrt, daß ein Business Process Reengineering nicht sinnvoll ist, wenn die bestehende Softwarelandschaft als Basis dienen soll. Auch sie muß reengineered oder neu entworfen und entwickelt werden.

Der letzte Punkt zeigt, daß mit dem Business Process Reengineering auch eine geeignete softwaretechnische Unterstützung durch die Informationssysteme des Unternehmens unabdingbar ist. Eine funktionszentrierte Abbildung von Prozeßfragmenten auf Informationssysteme, wie in Abbildung 2.3 dargestellt, ist nicht mehr sinnvoll.

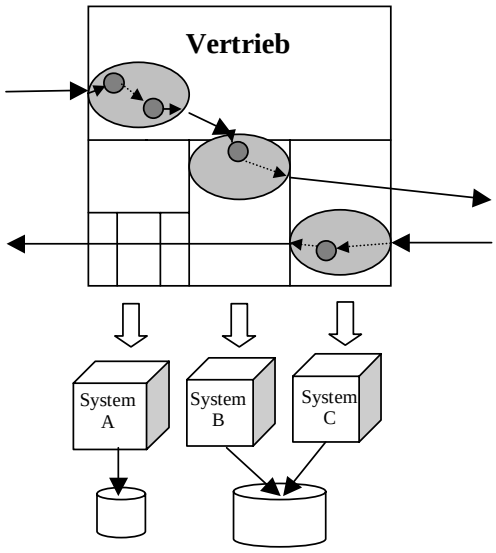

Abbildung 2.3: Funktionszentrierte Abbildung auf Informationssysteme

Doch gerade eine solche Vorgehensweise ist in vielen Unternehmen die gängige Praxis der Entwicklung von Informationssystemen. Schuld daran ist oftmals die Budgetpolitik. Die Erstellung, Wartung und der Betrieb von Informationssystemen muß von den einzelnen Fachabteilungen aus ihrem jeweiligen Budget bezahlt werden. Dementsprechend speziell sind auch die Anforderungen an diese Informationssysteme für die Unterstützung von Aufgaben. „Wer zahlt, schafft an“, oder anders ausgedrückt: Informationssysteme, die bereichsweite oder gar unternehmensweite Prozesse unterstützen, werden auf diese Weise nicht erstellt.

Aber auch die heutigen Software-Entwicklungsmethoden sind nicht dafür vorgesehen, einen unternehmensweiten Geschäftsprozeß als Ausgangspunkt für eine Entwicklung heranzuziehen. Zu sehr steht der Gedanke im Vordergrund, ein System zu entwickeln, das eine eigene Sichtweise auf die Prozesse des Unternehmens hat. Bezeichnend dafür sind die Use Cases, die bei mehreren Methoden [JGJ97, BRJ99] an den Anfang der Entwicklung gestellt werden. Die Use Cases beschreiben eine Systemgrenze, die Nutzer des Systems und die Anwendungsfälle, die das System unterstützt. Systemübergreifende oder unternehmensweite Sichten werden nicht bereitgestellt. Zudem sind Prozesse an sich, das heißt arbeitsteilige Vorgänge, in den „universellen“ Methoden nur sehr unzureichend modellierbar.

Es fehlt bei den traditionellen Methoden eine geeignete Unterstützung des Business Process Reengineering auf Seiten der Softwareentwicklung. Um zu klären, wie eine solche Unterstützung aussehen könnte, werden im nächsten Abschnitt

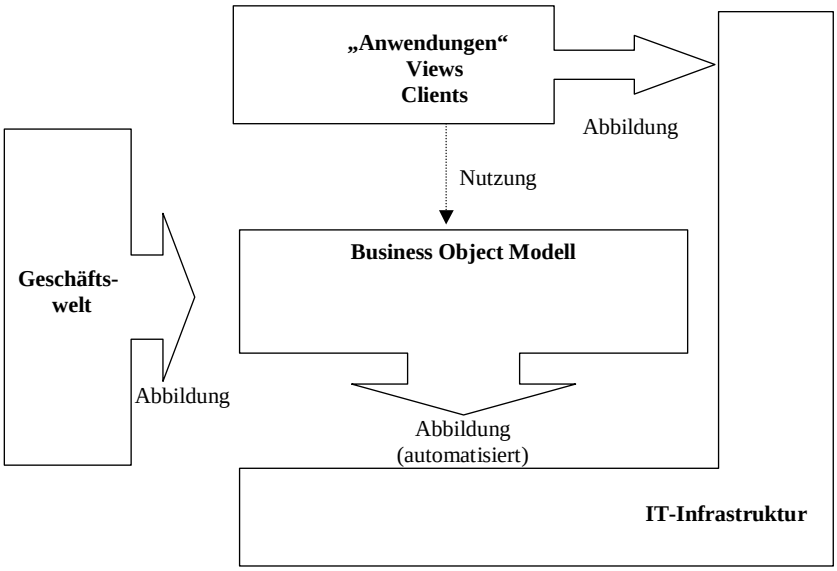

Abbildung 2.4: Modellierung der Geschäftswelt

die Anforderungen an die Softwaretechnik und das Softwareengineering diskutiert, die sich aus einer auf unternehmensweite Geschäftsprozesse ausgerichteten Entwicklung von Softwaresystemen – speziell Informationssystemen – ergeben. Auf dieser Basis wird ein methodisches Framework vorgeschlagen, das diese Anforderungen erfüllt.

2.2 Anforderungen

Hauptsächliches Problem ist es, eine geeignete Abbildung der Geschäftswelt auf Softwaresysteme zu finden, und zwar der Teile der Geschäftswelt, die durch Softwaresysteme unterstützt werden sollen.

Definition 2.3 (Geschäftswelt) *Die Geschäftswelt ist die Gesamtheit der* `Geschäftsprozesse` *mitsamt ihrer einzelnen Aktivitäten, Akteure und Produkte eines Unternehmens und seiner Partner (Kunden oder Zulieferer).*

Die Abbildung der Geschäftswelt ist kritisch für den Erfolg eines Business Process Reengineering: Spiegeln sich im Softwaresystem die an Geschäftsprozessen ausgerichtete Struktur der Geschäftswelt wider, so werden diese durch die Systeme auch optimal unterstützt. Wird die Geschäftswelt aber fragmentiert und

redundant auf eine heterogene Softwarelandschaft abgebildet, so ist der Erfolg eines Business Process Reengineering fraglich.

Definition 2.4 (Abbildung der Geschäftswelt) *Eine Abbildung der Geschäftswelt auf* `Informationssysteme` *erfolgt durch eine Modellierung der Elemente und Eigenschaften der* `Geschäftswelt` *und einer anschließenden Abbildung des* `Modells` *auf eine Software-Infrastruktur für Informationssysteme.*

Ziel wäre demnach eine Lösung, wie sie in Abbildung 2.4 dargestellt ist: Die Geschäftswelt, ihre Elemente, Strukturen und Abhängigkeiten, werden auf ein Modell abgebildet, dessen Architektur die Elemente, Strukturen und Abhängigkeiten widerspiegelt.

Definition 2.5 (Modell) *Ein Modell eines (realen)* `Systems` *beschreibt die für eine bestimmte Aufgabenstellung wesentlichen (abstrahierten) Elemente, Eigenschaften und Beziehungen des* `Systems`*.*

Definition 2.6 (System) *Unter einem System versteht man eine von seiner Umgebung abgegrenzte Anordnung aufeinander einwirkender Komponenten.*

In dieser Arbeit wird ein durch die Abbildung der Geschäftswelt entstandenes Modell Business Object Modell genannt. Business Objects sind die Abbildungen der Elemente der Geschäftswelt („Auftrag“, „Kunde“, „Vorgang Kundenreklamation“ usw.) mit wechselseitigen Beziehungen, die den Abhängigkeiten der abgebildeten Elemente der Geschäftswelt entsprechen.

Definition 2.7 (Business Object) *Ein Business Object ist eine* `Komponente` *eines* `Systemverbundes`*, die einem Element der* `Geschäftswelt` *entspricht. Die Abhängigkeiten und Verknüpfungen mehrerer Business Objects eines* `Systemverbundes` *untereinander entsprechen den Abhängigkeiten der von ihnen abgebildeten Elemente der* `Geschäftswelt`*.*

Zusätzlich zum Begriff Objekt als Abbildung eines Elementes der realen Welt in traditionellen, objektorientierten Methoden, werden Business Objects immer als Komponenten eines Systemverbundes betrachtet.

Definition 2.8 (Systemverbund (Informationssysteme)) *Ein Systemverbund (einer organisatorischen Einheit) besteht aus ein oder mehreren Informationssystemen, die verschiedene oder gleiche Teile der Geschäftsvorgänge (dieser organisatorischen Einheit) unterstützen.*

Definition 2.9 (Komponente (Software)) *Eine (Software-)Komponente ist ein Teil eines Softwaresystems, der durch Schnittstellen eindeutig spezifiziert ist und dessen Schnittstellen zur Laufzeit von anderen Komponenten des Systems aufrufbar sind.*

Objekte einer objektorientierten Programmiersprache sind in diesem Sinne im Allgemeinen keine Komponenten. Ein Softwaresystem selbst ist damit immer eine Komponente, da ein Softwaresystem ohne Schnittstelle zur Außenwelt keinen Sinn macht.

Die Einbindung des Modells in die Geschäftswelt selbst – also die Unterstützung durch Informationssysteme – erfolgt mithilfe der Anwendungen auf dieses Modell. Diese Anwendungen entsprechen traditionellen Informationsystemen mit einer Ausnahme: Die Funktionen, die das System im Rahmen seiner Unterstützung bieten soll, werden gemeinsam mit allen anderen Anwendungen durch die Nutzung und die Manipulation des Business Object Modells realisiert. Das Business Object Modell selbst ist jedoch völlig unabhängig von den Anwendungen.

Definition 2.10 (Business Object Modell) *Ein Business Object Modell umfaßt alle* `Business Objects` *und deren Verbindungen und Eigenschaften. Ein Business Object Modell enthält keine Abhängigkeiten zu Anwendungen oder Infrastrukturen.*

Ein Modell ist noch kein lauffähiges System. Dieses entsteht erst durch die Abbildung des Modells auf eine Business Object Infrastruktur.

Definition 2.11 (Business Object Infrastruktur) *Eine Business Object Infrastruktur ermöglicht die Realisierung und Ausführung von* `Business Objects` *als* `Komponenten` *eines* `Systems` *auf Computern.*

Die Infrastruktur muß neben einer geeigneten Unterstützung und Umsetzungsmöglichkeit der Elemente des Business Object Modells auch eine Nutzung des so

entstandenden Business Object Systems durch die Business Object Anwendungen ermöglichen.

Definition 2.12 (Business Object System) *Ein Business Object System ist eine (ausführbare) Abbildung eines* `Business Object Modells` *auf eine* `Business Object Infrastruktur`*. Ein Business Object System besteht aus* `Business Objects` *als eigenständige* `Komponenten` *auf die von anderen* `Systemen` *und* `Business Object Anwendungen` *in gleicher Form zugegriffen werden kann.*

Definition 2.13 (Business Object Anwendung) *Eine Business Object Anwendung ist ein* `Softwaresystem` *zur Einbindung des* `Business Object Systems` *in die* `Geschäftswelt`*. Im wesentlichen realisiert die Business Object Anwendung eine Benutzerschnittstelle zwischen Anwender und* `Business Object System`*. Eine Business Object Anwendung entwickelt kein eigenes Modell der Geschäftswelt, denn dies ist die Aufgabe des* `Business Object Systems`*, sondern realisiert nur die Anwendungslogik, um die* `Business Objects` *dem jeweiligen Benutzer darzustellen.*

Eine solche Lösung stellt Anforderungen an die Softwaretechnik und das Softwareengineering. Wichtige Anforderungen sind dabei Durchgängigkeit, Unterstützung der mehrfachen Verwendbarkeit der Business Objects, Offenheit und Technikunabhängigkeit. Was diese für ein geeignetes Vorgehen zur Entwicklung unternehmensweiter Systeme bedeuten, wird in den folgenden Abschnitten besprochen.

2.2.1 Durchgängigkeit

Die umfassende und ausschließliche Orientierung an den unternehmensweiten Geschäftsprozessen muß in allen Phasen des Entwicklungsprozesses eines unternehmensweiten Systems durchgängig verfolgt werden.

Definition 2.14 (Unternehmensweites Informationssystem) *Ein unternehmensweites Informationssystem ist ein* `Informationssystem`*, dessen Informationen Teil der* `Geschäftsprozesse` *einer Organisation sind.*

Es darf keinen Bruch zwischen den Modellen in Analyse und Design oder zwischen den Modellen in Design und Implementierung geben, weil sonst die entstehenden Systeme nicht die Struktur der Geschäftsprozesse widerspiegeln, die auf sie abgebildet werden. Deshalb müssen die gleichen Modellelemente und deren Beziehungen sowie die Sichten auf diese Modellelemente für alle Phasen des Prozesses im Mittelpunkt der Entwicklung stehen.

Die fehlende Durchgängigkeit strukturierter Methoden war der hauptsächliche Grund für die Entwicklung von objektorientierten Methoden. In der Praxis sind jedoch auch bei den objektorientierten Methoden noch immer Brüche zwischen den Phasen feststellbar, nämlich genau dann, wenn das fachliche Modell im Design und noch stärker in der Implementierung mit Architektur- und Infrastrukturelementen verknüpft wird und dabei die ursprüngliche Struktur mehr und mehr in den Hintergrund tritt.

2.2.2 Unterstützung der Mehrfachverwendung

in dieser Arbeit wird in Abgrenzung zu den Arbeiten zur Wiederverwendung von Mehrfachverwendung und Mehrfachnutzung gesprochen. Klassifiziert werden diese durch unterschiedliche Ausprägungen dreier Charakteristika: Der Intention bei der Erstellung, das Prozeßstadium und die Laufzeitnutzung der wieder oder mehrfach zu verwendenden Produkte. Die Arbeit fokussiert sich im Folgenden auf das Produkt Business Object.

Bei der Intention bei der Erstellung eines wieder- oder mehrfach zu verwendenden Produktes kann man zwei Ausprägungen unterscheiden: Das Produkt wird mit der Intention erstellt für den gleichen Zweck mehr als einmal verwendet zu werden oder nicht. Ausschlaggebend ist hierbei die gezielte Planung der mehrmaligen Verwendung eines Produktes in seiner Planungs- und Implementierungsphase: Ein Datenbankserver wird schon von vornherein so geplant und entworfen, daß der von vielen unterschiedlichen Anwendungen zum gleichen Zweck, nämlich der persistenten Speicherung von Daten in relationaler Form, verwendet werden kann. Eine (Ad-hoc-)Wiederverwendung von Software ohne vorherige Planung bedingt in der Regel ein aufwendiges Reengineering.

Das Merkmal Prozeßstadium bezeichnet, in welchem Stadium eines Entwicklungsprozesses das entsprechende Produkt als Ergebnis erarbeitet wurde. Es wird hier von zwei hauptsächlichen Ausprägungen für dieses Merkmal ausgegangen. Entweder, das Produkt ist Ergebnis der Planungs- und Implementierungsphase, so wie ein Algorithmus oder Teile eines Quelltextes eines Programms oder es ist Er-

gebnis der Produktionsphase, so wie eine kompilierte Klassenbibliothek oder das fertige, ausführbare Programm.

Bei der Entwicklung von Software tritt jedoch bei der geplanten, mehrmaligen Verwendung oft ein komplexerer Fall ein: Zwei zusammenhängende Produkte jeweils aus einerseits der Planungs- und Implementierungsphase und andererseits der Produktionsphase werden gemeinsam verwendet. Zum Beispiel bei der Nutzung einer objektorientierten Klassenbibliothek: Man nutzt neben den bereits kompilierten Bibliotheken als Ergebnis der Produktionsphase die dazugehörigen Klassendefinitionen als Ergebnis der Planungs- und Implementierungsphase. Ohne diese Klassendefinitionen könnte das Produkt „Klassenbibliothek“ nicht in die Anwendung eingebunden werden. Dasselbe gilt für Betriebsystem-Bibliotheken oder Frameworks zur Anwendungsentwicklung.

Zur Erweiterung des Merkmals Prozeßstadium wird ein drittes Merkmal betrachtet, die Laufzeitnutzung bei Produkten aus der Produktionsphase. Während die beiden obengenannten Merkmale auch für die mehrmalige Verwendung von klassischen Industrieprodukten eine Rolle spielen, beschreiben die Ausprägungen Instanz und Kopie der Eigenschaft Laufzeitnutzung ein Merkmal, das nur aufgrund des immateriellen Charakters des Produktes Software eine Rolle spielt. Kopie bedeutet, daß das Produkt (eine Bibliothek o.ä.), falls es mehrmals zur Laufzeit verwendet wird, pro Verwendung jeweils eine eigene Instanz mit einer eigenen Identität und eigenem (Daten-)Zustand darstellt. Hierbei spielt es keine Rolle, ob diese Verwendung zeitlich parallel oder sequentiell erfolgt. Instanz bedeutet, daß bei mehrmaliger Verwendung des Produkts (eine Datenbank o.ä.) zur Laufzeit immer mit einem gemeinsamen Zustand gearbeitet wird.

Damit ergeben sich drei verschiedene Strategien:

Definition 2.15 (Wiederverwendung) *Von Wiederverwendung spricht man, wenn ein Produkt (Quelltext, Modelle o.ä.) für die Entwicklung einer Software verwendet wird und keine Intention bei der Erstellung des Produktes für eine solche erneute Verwendung vorhanden war.*

Beispiele sind die Wiederverwendung von Programm-Quelltexten (Algorithmen, Unterprogramme) oder die Wiederverwendung eines Architekturmodells, die in [Kru92] als Code- beziehungsweise Design-Scavenging bezeichnet werden.

Definition 2.16 (Mehrfachverwendung) *Wird ein Produkt (meist eine Kombination aus zwei Produkten aus der Planungs-/Implementierungsphase und der*

Produktionsphase), das schon von vornherein mit der Intention entwickelt wurde mehrmals verwendet zu werden, für die Entwicklung einer Software verwendet, so spricht man von Mehrfachverwendung.

Definition 2.17 (Mehrfachnutzung) *Mehrfachnutzung ist ein spezieller Fall der Mehrfachverwendung im Rahmen eines Systemverbundes (siehe Definition 2.8). Die Strategie Mehrfachnutzung sieht vor, daß es zur Laufzeit nur einen gemeinsamen Zustand des jeweils für die Entwicklung der am Systemverbund beteiligten Systeme mehrfach verwendeten Produktes gibt. Produkte, die mehrfach genutzt werden können, können demnach (in voneinander unabhängigen Systemen) auch mehrfach verwendet werden aber nicht umgekehrt.*

Sollen viele verschiedene Anwendungen auf einem Business Object System arbeiten, so müssen dessen Elemente – die Business Objects – mehrfachnutzbar sein. Business Objects müssen integrierte, in allen Unternehmensteilen verwendbare Elemente der Geschäftswelt für die Anwendungen repräsentieren, die zur Laufzeit im Systemverbund von anderen Business Objects und Anwendungen genutzt werden können.

Dabei hat die Unterstützung der Mehrfachnutzung der Business Objects ein viel größeres Gewicht als die Mehrfachverwendung während der Entwicklung eines Systems. Gerade der letzte Punkt wird in heutigen Methoden stark betont. Die verstärkte Nutzung von komponentenorientierten Technologien und Methoden ist ein Kennzeichen hierfür. Für nicht-fachliche Komponenten, wie eine GUI-Komponente oder einen Autorisierungsmanager, ist die Nutzung von komponentenorientierten Techniken von Vorteil, da die Komponenten bei der Entwicklung von vielen Systemen systematisch und damit kostensparend mehrfachverwendet werden können. Doch solange man keine unternehmensübergreifenden Lösungen entwickelt, ist der Nutzen einer Mehrfachverwendung von Business Objects zur Entwicklungszeit fraglich, da ein Element der Geschäftswelt auf ein Business Object abgebildet und damit auch nur ein Mal entwickelt wird.

Wichtig bei der Mehrfachnutzung der Business Objects ist einerseits eine geeignete Infrastruktur zur Bereitstellung von verteilten Objekten und andererseits eine möglichst feingranulare Partitionierung und Realisierung dieser Komponenten. Die Partitionierung und Granularität muß sich an den Elementen der Geschäftswelt orientieren. Erlaubt die Infrastruktur nur größere, grobgranulare Komponenten, entsteht das Problem des „architectual mismatch“ [GAO95] zwischen den Komponenten und den Business Objects.

Die Verknüpfung von Business Objects und anwendungsspezifischer Modelle

muß unbedingt vermieden werden. Dies schränkt die Mehrfachverwendung stark auf einen Anwendungsfall ein und verhindert die Mehrfachnutzung solcher Business Objects durch andere Anwendungen.

2.2.3 Offenheit

Definition 2.18 (offenes Informationssystem) *Ein offenes Informationssystem besteht aus* `Komponenten`*, deren Schnittstellen sowohl innerhalb des* `Informationssystems`*, als auch von anderen* `Systemen` *(in gleicher Art und Weise) genutzt werden können.*

Die Offenheit eines Business Object Systems ist notwendig für eine umfassende, unternehmensweite Umsetzung des Business Reengineering Ansatzes auf die Softwaresysteme. Denn ein Business Process Reengineering ist nur dann erfolgreich, wenn es im Unternehmen keine Systeminseln und keine heterogene und wenig integrierte Softwarelandschaft gibt.

Dies kann nur dadurch erreicht werden, daß die Business Objects von einer Vielzahl unterschiedlicher Anwendungen genutzt und manipuliert werden können. Ein wenig offenes Business Object System birgt die Gefahr, daß Anwendungen aufgrund der Schwierigkeiten bei der Anbindung am Business Object System vorbei entwickelt werden und damit wieder eine redundante Sicht der Geschäftswelt darstellen.

2.2.4 Technikunabhängigkeit

Die Methode zur Entwicklung von Business Object Systemen sollte, vor allem bei dem Entwurf des Business Object Modells, möglichst alle technischen Abhängigkeiten ausblenden. Denn die Entwicklung und Nutzung eines Business Object Modells verläuft über einen sehr langen Zeitraum, unter Umständen über die gesamte Lebensdauer des Unternehmens. IT-Technologien und -Infrastrukturen haben verglichen damit jedoch einen sehr kurzen Entwicklungs- und Einsatzzyklus. Auf CORBA folgen Enterprise JavaBeans, auf Enterprise JavaBeans folgen .NET-Komponenten und all das in einem sehr kurzen Zeitraum.

Deshalb ist es für ein unternehmensweites Vorgehen wichtig, innovationsoffen zu sein. Das heißt im Idealfall: Die verwendete Infrastruktur läßt sich ohne Änderung des Modells austauschen. Dies kann nur über einen generativen Ansatz erreicht

werden, bei dem das Business Object System möglichst automatisch mithilfe von Abbildungsregeln für eine gewählte Infrastruktur generiert wird. Will man die Infrastruktur austauschen, so sind nur der Austausch des Generator-Backends und neue Abbildungsregeln notwendig.

2.3 Übersicht über das methodische Framework

Kern dieser Arbeit ist die Erarbeitung eines methodischen Frameworks zur Entwicklung von unternehmensweiten Systemen auf der Basis von Business Objects. In diesem Abschnitt wird dieses Framework in der Übersicht dargestellt und es wird diskutiert, inwieweit die oben genannten Anforderungen erfüllt wurden.

Ziel bei der Entwicklung eines unternehmensweiten Vorgehens war eine erweiterbare und dennoch integrierte Lösung von Konzepten aus verschiedenen Bereichen: Modellierung, Vorgehensmodell, Entwicklungswerkzeuge und Abbildung auf eine Infrastruktur. Deshalb wurde ein Framework-Ansatz gewählt: Die Lösungen in den Bereichen bauen auf einer gemeinsam genutzten, integrierenden Frameworkbasis auf – dem Business Object Metamodell.

Die Lösungen sind als eine mögliche Ausprägung eines unternehmensweiten Vorgehens zu verstehen und können durch eine andere Lösung ersetzt werden, ohne die anderen Teile des Frameworks obsolet werden zu lassen. Auch erheben die Lösungen keinen Anspruch auf Vollständigkeit; jedoch wurde jeder Bestandteil des Frameworks so konzipiert, daß Erweiterungen zur Anpassung an die Bedürfnisse des jeweiligen Unternehmens beziehungsweise der jeweiligen Branche möglich sind.

2.3.1 Metamodell

Definition 2.19 (Metamodell) *Ein Metamodell beschreibt eine Klasse beziehungsweise eine Menge von Modellen. Es legt fest, welche Elemente, Eigenschaften und Beziehungen eines* `Systems` *durch ein* `Modell` *beschrieben werden können. Ein Metamodell definiert und sichert ferner die Einhaltung von Regeln auf den möglichen Elementen eines auf Basis des Metamodell instanziierten* `Modells`*.*

In dieser Arbeit ist entspricht im Business Object Metamodell das System der abzubildenden Geschäftswelt und das Modell dem Business Object Modell. Das

Business Object Metamodell ist Basis und Integrationspunkt des Frameworks. Mit ihm werden die möglichen Elemente und Abhängigkeiten eines Business Object Modells festgelegt. Es legt zudem die Granularität und die möglichen Schnittstellen der Elemente fest. Damit bietet es eine Grundlage für eine systematische Abbildung und Wiederverwendung der Elemente der Geschäftswelt.

Das Business Object Metamodell sieht keine Elemente zur Modellierung von technischen oder anwendungsabhängigen Elementen vor. Dadurch wird eine durchgängige und technikunabhängige Lösung für die anderen Bestandteile vorbereitet und erzwungen.

Wichtig für die konsistente Abbildung der Geschäftswelt ist die Integration von Elementen zur Modellierung von arbeitsteiligen Vorgängen. Das Metamodell bietet hierzu eine durchgängige und offene Lösung, ohne von einem konkreten Workflow-Management-Werkzeug oder -Modellierungs-Werkzeug abhängig zu sein.

2.3.2 Modellierung

Für die Modellierung des Business Object Modells wird ein Profil der UML auf Basis des Business Object Metamodells definiert. Dieses Profil bietet einen ausreichenden Satz an Beschreibungstechniken zur Modellierung aus unterschiedlichen Sichten.

Durch die Abbildung auf das Business Object Metamodell sind die einzelnen Beschreibungstechniken integriert und können ohne konzeptionelle Brüche phasenübergreifend verwendet werden.

Die Erweiterung der UML wird durch spracheigene Erweiterungsmittel (Stereotypen, Tags, Constraints) erreicht. Die Zielvorgabe ist dabei, daß kommerzielle UML-Modellierungswerkzeuge zur Bearbeitung des Business Object Modells genutzt werden können.

2.3.3 Vorgehensmodell

Definition 2.20 (Vorgehensmodell) *In der Softwaretechnik beschreibt ein Vorgehensmodell die Abwicklung eines Projektes zur Erstellung eines Softwaresystems. Die Abwicklung wird durch die Definition von durchzuführenden Tätigkeiten und zu erstellender Produkte in einzelne Phasen und Schritte unterteilt. Dabei werden*

die möglichen Sichten auf und die jeweiligen Modelle des Softwaresystems den Produkten zugeordnet und miteinander verzahnt. Ein Vorgehensmodell setzt ferner die zu erstellenden Produkte in Beziehung zueinander und beschreibt deren Abhängigkeiten untereinander.

Durch die Trennung von unternehmensweiten Business Object Systemen und den eigentlichen Anwendungen ist ein paralleles und verzahntes Vorgehen bei der Entwicklung von Business Object System und Business Object Anwendung notwendig. Auch unterscheiden sich die Rollen und Aktivitäten bei der Entwicklung von Business Object Systemen von denen bei der Entwicklung traditioneller Informationssysteme.

Die Lösung für das Framework ist die Erweiterung und Anpassung eines bestehenden Prozeßframeworks, dem Unified Process, durch die Angabe von neuen Rollen, Aktivitäten und Produkten für die Workflows des Prozeßframeworks.

2.3.4 Werkzeuge zur Entwicklung unternehmensweiter Systeme und Anwendungen

Ein wichtiger Erfolgsfaktor für ein unternehmensweites Vorgehen auf der Basis von Business Objects ist eine geeignete Werkzeugumgebung. Als abschließender Bestandteil des methodischen Frameworks wird eine dreischichtige Architektur einer Werkzeugumgebung vorgeschlagen. Dabei werden Modell, Infrastrukturanpassung und Abbildung unterschieden.

Für die Modellierung wird ein Werkzeugkonzept basierend auf dem kommerziellen UML-Werkzeug Together vorgestellt. Für die Anpassung an eine Infrastruktur wird eine Sprache zur Beschreibung eines Infrastrukturmodells entwickelt, die eXtendet Business Object Definition Language (XBODL). Und für die Abbildung wird eine geeignete Infrastruktur, das Enterprise JavaBeans Framework, ausgewählt und die Regeln für die Generierung der Business Object Modellelemente auf Enterprise JavaBeans angegeben.

Business Object Systeme, die mithilfe dieser Werkzeugumgebung erstellt werden, sind offene, von vielen Anwendungen nutzbare, verteilte Komponentensysteme.

2.4 Zusammenfassung

Durch die fundamentale Neuausrichtung der Unternehmensstrukturen auf Kundengetriebene Geschäftsprozesse und die Auflösung der traditionellen organisatorischen Grenzen durch das Business Process Reengineering, werden neue Anforderungen an den Aufbau und die Entwicklung von Informationssystemen gestellt.

Konventionelle Architekturen und Methoden sind für die Entwicklung solcher Systeme nicht geeignet, da sie die Anforderungen an die Durchgängigkeit, Wiederverwendbarkeit, Offenheit und Technikunabhängigkeit von unternehmensweiten Systemen nur unzureichend berücksichtigen.

In dieser Arbeit wird deshalb ein methodisches Framework zur Entwicklung von offenen, unternehmensweiten Anwendungen auf der Basis von Business Objects entwickelt. Basis des Frameworks ist ein Business Object Metamodell. Im nächsten Teil der Arbeit werden die Grundlagen für das Metamodell vorgestellt: Der Business Object Ansatz, Möglichkeiten zur Integration von Workflows und die in dieser Arbeit für das Metamodell getroffenen Design-Entscheidungen.

Kapitel 3

Business Objects

Business Objects sind das grundlegende Konzept des in dieser Arbeit entwickelten methodischen Frameworks. Sie sind die Basis für den Entwurf und die Entwicklung offener, unternehmensweiter Informationssysteme. In diesem Kapitel werden die Eigenschaften von Business Objects, die Möglichkeiten zur deren Modellierung und mögliche Business Objects Architekturen zusammen mit aktuellen Arbeiten vorgestellt und diskutiert.

Die Diskussion führt zu der Aufstellung von grundlegenden Kriterien für ein auf Business Objects basierenden Vorgehen zur Entwicklung unternehmensweiter Systeme.

3.1 Eigenschaften

Wie in den Definitionen 2.7 (Business Object) und 2.12 (Business Object System) dargestellt ist der Kerngedanke bei einem Business Objects Ansatz, daß Business Objects in allen Entwicklungsphasen eigenständige Elemente eines Informationssystems sind, auf die die Elemente der Geschäftswelt in gleichartiger Form abgebildet und miteinander verbunden werden. Finden sich in der Geschäftswelt Elemente wie „Auftrag“ oder „Artikel“, so werden für das Informationssystem die Business Objects „Auftrag“ und „Artikel“ definiert, entworfen und implementiert und sind innerhalb und außerhalb des Informationssystems in dieser Form zugreifbar.

Hier unterscheidet sich das Konzept vom traditionellen, objektorientierten Vorge-

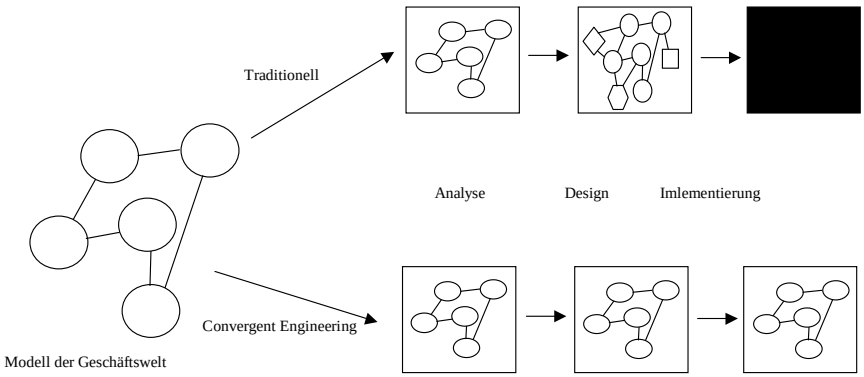

Abbildung 3.1: Traditionelles und Convergent Engineering

hen. Auch hier wird in der Analysephase ein fachliches Objektmodell erstellt, das ebenfalls eine Abbildung von Elementen der Geschäftswelt darstellt. Allerdings wird dann im Design und während der Implementierung keine Unterscheidung gemacht zwischen diesen Objekten und technischen Objekten, die für den Aufbau der GUI, dem Sitzungsmanagement oder der Speicherung von Daten zuständig sind.

Zudem können die Objekte klassischer objektorientierter Systeme zur Laufzeit nicht von anderen Anwendungen mehrfach genutzt werden. Das entstandene Objektgeflecht und die monolithische Anwendung zur Laufzeit entsprechen nicht mehr oder nur noch eingeschränkt den Elementen der Geschäftswelt. In Abbildung 3.1 werden die Unterschiede zwischen einem Informationssystem basierend auf Business Objects und einem traditionellen Informationssystem schematisch dargestellt.

Die Integration einer Softwarelandschaft und die Offenheit der Informationssysteme (siehe Definition 2.18 (offenes Informationssystem)), die auf Business Objects basieren soll durch das Konzept der gleichartigen Abbildung von selbst erreicht werden: Die Integration geschieht dadurch, daß es die einzelnen Arten von Business Objects wie „Auftrag“ oder „Artikel“ analog zur Geschäftswelt nur einmal in der Softwarelandschaft eines Unternehmens gibt. Alle IT-unterstützten Abläufe oder Prozesse, die mit der Verarbeitung dieser Geschäftselemente zu tun haben, setzen auf eine gemeinsame, unternehmensweite Basis von Business Objects auf. Dadurch soll auch ein möglichst hoher Grad an Mehrfachverwendung fachlicher Objekte erreicht werden. Dabei kann man, wie in der Einleitung bereits skizziert, zwei Arten der Mehrfachverwendung unterscheiden: Unternehmensweite- und Unternehmensübergreifende Mehrfachverwendung von Business Objects.

Unternehmensweite Mehrfachverwendung ist in der Regel eine Mehrfachnut-

zung (siehe Definition 2.17 (Mehrfachnutzung)). Von Anwendungen unabhängige Business Objects werden zur Laufzeit von mehreren Anwendungen benutzt. Unternehmensübergreifende Mehrfachverwendung geschieht meist bei der Konstruktion eines Systemtyps, wie zum Beispiel die Mehrfachverwendung des Business Objects „Auftrag“ für Auftragsverwaltungssysteme. Die vorliegende Arbeit fokussiert sich wie in der Einleitung bereits angesprochen, aufgrund der Schwierigkeit zwei Verhaltensbeschreibungen miteinander zu vergleichen, auf eine unternehmensweite Mehrfachverwendung von Business Objects.

Die ersten Ideen und Konzepte für Business Objects entstanden bereits 1994 [SLJR94, Sim94, Tay94]. Seit 1995 beschäftigt sich eine eigens zu diesem Zweck gegründete Business Objects Domain Task Force (BODTF) der Object Management Group (OMG) mit dem Thema. Parallel dazu findet seit 1995 jährlich ein internationaler Workshop „Business Objects Design and Implementation“ statt [SPC+97, PSM98, PSM99]. Auch sind zahlreiche Publikationen erschienen [ES98, Pri96, AF98, HS99]. Die Arbeiten zu Business Objects werden in dieser Arbeit in drei Themengebiete eingeteilt: Modellierung, Architektur und Realisierung. Sie werden im folgenden näher betrachtet.

3.2 Modellierung

Wenn mit Business Objects offene, unternehmensweite Informationssysteme (siehe Definitionen 2.18 (offenes Informationssystem) und 2.14 (unternehmensweites Informationssystem)) erstellt werden sollen, so ist die einheitliche Modellierung von Business Objects eine wichtige Voraussetzung. Da die Modellbildung einer Situation in der Geschäftswelt im allgemeinen ein kreativer Akt ist, gelangt man ohne ein standardisiertes Framework aus geeigneten Modellelementen nicht zu einem einheitlichen Ergebnis. Unterschieden werden muß hierbei zwischen den Metamodellen von Business Objects – in der Literatur häufig auch Klassifizierung, Taxonomie oder Typisierung genannt – und den Referenzmodellen von Geschäftsbereichen. Zudem werden Business Objects nach ihrer Spezialisierung auf ein bestimmtes Unternehmen oder Generalisierung für eine ganze Branche in verschiedene Spezialisierungsstufen klassifiziert.

3.2.1 Business Object Metamodell

Wie in der Einleitung schon erwähnt, gibt es eine große Übereinstimmung in den Arbeiten zu Business Object Metamodellen, daß zumindest zwischen Business Entity Objects und Business Process Objects unterschieden werden sollte. Darüber hinaus gibt es allerdings keinen Konsens. Auch werden teilweise Modellierungsaspekte mit denen im nächsten Abschnitt besprochenen Architekturaspekten vermischt, wie zum Beispiel die Einführung eines Subsystem Business Objects in der BOCA der OMG [OMG98a]. Andere Modellelemente sind zum Beispiel Business Organisation Objects [Tay94, Mar00], die eine betriebliche Organisationsstruktur abbilden sollen, Business Purpose Objects [Mar00], die den wirtschaftlichen Grund, der hinter den im Unternehmen durchzuführenden Prozesse steht, darstellen oder Business Utility Objects [HS99], die allgemeine fachliche Konzepte, wie „Adressbuch“ oder „Währung“ gruppieren.

Neben Business Entity Objects und Business Process Objects werden am häufigsten Business Event Objects und Business Rule Objects als weitere Modellelemente eingeführt. Abbildung 3.2 zeigt ein oft zitiertes Modell [She97] mit weiteren Verfeinerungen der Grundtypen. Business Event Objects stellen inner- und außerbetriebliche Ereignisse dar, wie zum Beispiel „Kunde ruft an“ oder „Lagerbestand unter Soll“. Business Rule Objects bilden Regeln, Standards oder Gesetze ab, nach denen Abläufe in einem Unternehmen durchzuführen sind.

3.2.2 Referenzmodelle für Business Objects

Bei der Entwicklung von Referenzmodellen für Geschäftsbereiche werden Lösungen vor allem unter dem Aspekt der Interoperabilität von Anwendungen entwickelt. Schon einige Zeit existieren mit EDI [Sok95] und CDIFF [Ern98] Formate, die es erlauben fachliche Daten unternehmensübergreifend auszutauschen. In der Object Management Group (OMG) gibt es verschiedene Arbeitsgruppen, die Objektmodelle entwickeln, die über den reinen Datenaustausch hinausgehen. Die sogenannten Domain Task Forces erarbeiten dabei Objektmodelle für verschiedene Anwendungsbereiche, wie zum Beispiel der Medizin-, Produktions- oder Telekommunikationsbranche [OMG95]. Dabei handelt es sich um Schnittstellenbeschreibungen der fachlichen Objekte, die über diese Schnittstellen auch zur Laufzeit eines Anwendungsprogramms auf Basis dieses Objektmodells zugreifbar sind.

Leider werden die Prozesse, die diese Objekte erschaffen, nutzen und verwalten

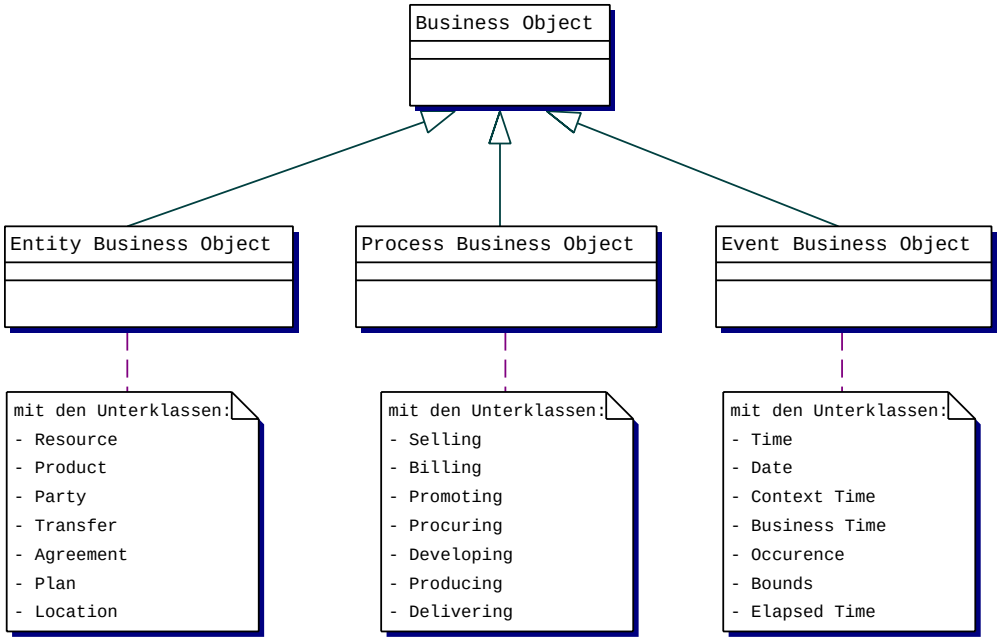

Abbildung 3.2: Sheltons Business Object Metamodell

nicht definiert. Deshalb können diese Modelle zwar unter Umständen als Referenzmodelle für Business Entity Objects verwendet werden, nicht jedoch für Business Process Objects. Gleiches gilt für die Business Object Documents der Firma SAP [TM97], die ebenfalls nur Schnittstellenbeschreibungen von Business Entity Objects darstellen.

3.2.3 Spezialisierungsstufen von Business Objects

In vielen Arbeiten werden Gültigkeitsbereiche für Business Objects definiert, je nachdem wie spezialisiert bzw. generalisiert sie sind. Abbildung 3.3 zeigt eine von der OMG [OMG95] vorgeschlagene Hierarchie.

Die Spezialisierungsstufen sind dabei:

- Common Business Objects (CBO): Sie stellen fach- und branchenneutrale Konzepte der Geschäftswelt dar, wie zum Beispiel „Termin“ oder „Währung“. Die OMG erarbeitet Spezifikationen solcher CBOs [OMG95]. Die bisherigen Erfahrungen zeigen jedoch, daß es sehr schwierig ist, sich auf eine allgemeingültige Festlegung von Schnittstellen zu einigen [OMG95].

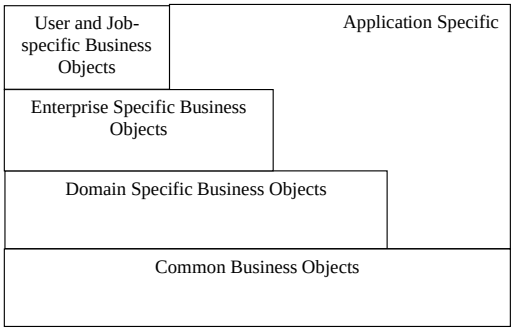

Abbildung 3.3: Spezialisierungshierarchien

- Domain Specific Business Objects: Sie bilden fach- oder branchenspezifische und damit unternehmensübergreifende Konzepte der Geschäftswelt ab. Dabei stützen sie sich auf CBOs. Domain Specific Business Objects könnten nach den oben erwähnten Referenzmodellen spezifiziert werden. Voraussetzung wären, daß sie mithilfe eines einheitlichen Metamodells (siehe Abschnitt 3.2.1) modelliert sind, sich auf CBOs abstützen und neben den Informationselementen auch die Prozesse spezifizieren. Wie oben schon erwähnt, wird die letzte Voraussetzung von keinem bisher verfügbaren Referenzmodell erfüllt.

- Enterprise Specific Business Objects: Entsprechen die CBOs oder Domain Specific Business Objects nicht den Abläufen einer Unternehmung und müssen dementsprechend angepaßt oder müssen zusätzliche Objekte definiert werden, so spricht man von Enterprise Specific Business Objects. Dabei kann man diese weiter unterteilen in Enterprise Specific Common Business Objects, also Business Objects, die Grundlage für eine Vielzahl von Prozessen einer Unternehmung sind und Enterprise and Domain Specific Business Objects, also Business Objects, die nur in bestimmten Anwendungsfeldern wie Finanzen, Marketing oder Produktion eine Rolle spielen.

- User and Job-specific Business Objects: Sie sind die am weitesten spezialisierten Business Objects und werden nur im Kontext spezieller Nutzer- oder Vorgangsanforderungen verwendet. User and Job-specific Business Objects bilden die Grenze dessen was unter Business Objects verstanden werden sollte. Eine weitere Spezialisierung zum Beispiel aufgrund von Anforderungen der Anwendung widerspricht der eigentlichen Definition von Business Objects. Solche Objekte werden auch Application Objects genannt.

3.3 Architektur

Neben einer einheitlichen Modellierung von Business Objects müssen auch die Architekturen der Systeme, die auf Business Objects basieren, vereinheitlicht werden. Softwarearchitekturen werden bestimmt durch funktionale wie nicht-funktionale Anforderungen.

Definition 3.1 (Funktionale Anforderungen) *Funktionale Anforderungen bestimmen, welche Aufgaben ein Softwaresystem in seiner jeweiligen Umgebung zu erfüllen hat. Dies ist die Beschreibung der Interaktion zwischen dem System und seiner Umgebung, also die Beschreibung des Verhaltens des Systems als Reaktion auf externe Stimuli.*

Definition 3.2 (Nicht-funktionale Anforderungen) *Nicht-funktionale Anforderungen bestimmen, welche Eigenschaften die Umsetzung der* `funktionalen Anforderungen` *auf die IT-Infrastruktur haben muß. Sie schränken die Wahlmöglichkeiten für eine solche Umsetzung für den Entwickler des Systems ein.*

Definition 3.3 (Softwarearchitektur) *Eine Softwarearchitektur umfaßt ein* `Modell` *der Software- und System-*`Komponenten` *mit deren Verbindungen, Rollen und darauf wirkende Einschränkungen, eine Sammlung von* `nicht-funktionalen Anforderungen` *sowie die Designentscheidungen, die zeigen, welche Auswirkungen die* `nicht-funktionalen Anforderungen` *auf das oben genannte* `Modell` *haben.*

Traditionelle Anwendungen haben stets eigenen funktionalen Anforderungen, daß heißt ihr eigenes Modell der Geschäftswelt, selbst wenn sie über eine Datenbank mit anderen Anwendungen integriert sind. Business Objects stellen jedoch ein gemeinsames Modell in einer Anwendungsarchitektur für viele Anwendungen dar. Daß heißt, die funktionalen Anforderungen aller Anwendungen beeinflussen die Architektur eines Business Object Systems.

Getrennt davon zu betrachten sind die nicht-funktionalen Anforderungen an die Architektur zur Ausführung, Verwaltung und Kooperation von Business Objects im System. Geprägt durch [OMG98a] wird hierfür oft der Begriff Business Object Facility gebraucht, hier Business Object Ausführungsarchitektur. Schließlich ist noch eine Architektur anzugeben, wie die Business Objects selbst aufgebaut sind. Diese hängt von den Vorgaben der Ausführungsarchitektur ab. Dazu wird der Begriff Business Object Instanzarchitektur eingeführt. Die Arbeiten zu den drei Architekturen werden im Folgenden vorgestellt.

3.3.1 Business Object Anwendungsarchitektur

Definition 3.4 (Business Object Anwendungsarchitektur) *Eine Business Object Anwendungsarchitektur sieht drei getrennte Schichten vor: Eine Schicht mit dem* `Business Object System`*, eine Schicht mit den* `Systemen` *zur Verwaltung und Speicherung der Informationen des* `Business Object Systems` *und eine Schicht für die* `Business Object Anwendungen`*. Letztere ist abhängig von der Schicht mit den* `Business Object System` *und die* `Komponenten` *dieser* `Schicht` *greifen auf die* `Komponenten` *des* `Business Object Systems` *zu, aber nicht umgekehrt.*

Mit Business Objects soll ein Modell der Geschäftswelt vielen Anwendungen im Unternehmen zur Laufzeit zur Verfügung stehen. Dieses Vorgehen erzwingt eine bestimmte Architektur und eine neue Art von Anwendungen. Geeignete Architekturen sind mehrschichtige (multi-tier) Architekturen [Hir96]. In diesen Architekturen werden unternehmensweite Anteile, die Business Objects, in einer eigenen, mittleren Schicht (middle-tier) realisiert, die zwischen davon getrennten Schichten einerseits zur Präsentation und Nutzerinteraktion, andererseits zur Speicherung und Verwaltung von Informationen liegt.

Dadurch ergibt sich auch eine neue Art von Anwendungen: Während bisher Anwendungen von der Darstellung über das eigene Modell bis zur Speicherung alleine für alles zuständig waren, sind Anwendungen, die auf einen gemeinsamen Verbund von Business Objects zugreifen nur noch zuständig für die Umsetzung einer bestimmten Sichtweise auf eine Teilmenge der unternehmensweiten Schicht Dadurch einsteht die in Abbildung 3.4 gezeigte dreischichtige (three-tier) Grundarchitektur.

In der Literatur wird diese Grundarchitektur allgemein akzeptiert [AF98, BJNR98, ES98, HS99, Mar00, SPC+97, PSM98, PSM99]. Je nach dem welche Spezialisierungsstufen aus Abschnitt 3.2.3 das jeweilige Business Object Modell umfaßt, können diese auch jeweils eigene Schichten statt einer mittleren Schicht in der Architektur darstellen. Allen und Frost [AF98] schlagen eine generelle Trennung der mittleren Schicht in unternehmensweite und lokale (abteilungs-, anwendungsfall- oder nutzerspezifische) Anteile vor.

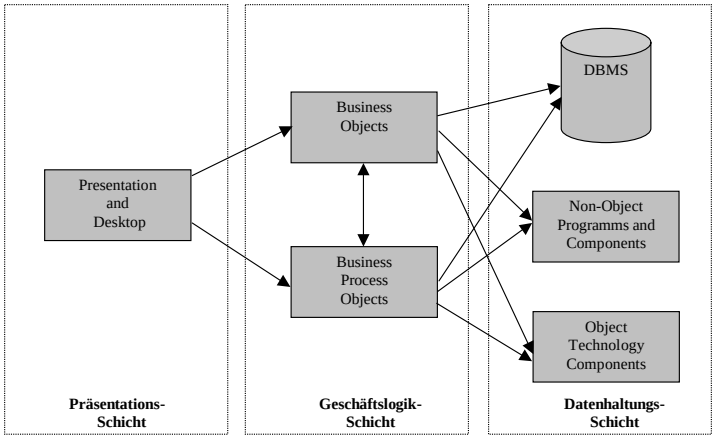

Abbildung 3.4: Dreischichtige Grundarchitektur (nach [OMG94])

3.3.2 Business Object Ausführungsarchitektur

Definition 3.5 (Business Object Ausführungsarchitektur) *Die Business Objects Ausführungsarchitektur ist bestimmt durch die Konzepte und Möglichkeiten des* `Business Object Metamodells`*. Ein* `Business Object Modell` *als Instanz eines* `Business Object Metamodells` *bestimmt die möglichen Objektgeflechte aus Instanzen der modellierten* `Business Objects` *und deren Assoziationen untereinander. Als Business Object Ausführungsarchitektur wird ein solches Objektgeflecht bezeichnet.*

Um Business Objects nach dem Prinzip der gleichartigen Abbildung von Geschäftsmodell in das Softwaremodell als Komponenten eines Informationssystems in einer standardisierten Art und Weise ablaufen und miteinander kooperieren zu lassen, muß eine Business Object Ausführungsarchitektur vorgegeben sein. Eine solche Ausführungsarchitektur basiert auf einem Business Object Metamodell (siehe Abschnitt 3.2.1) und hat zwei hauptsächliche nicht-funktionale Anforderungen zu erfüllen:

- Interoperabilität: Business Object Systeme bestehen in der Regel aus mehreren Business Objects. Auch ist es wahrscheinlich, daß Business Objects mit unterschiedlichen Werkzeugen, auf Basis unterschiedlicher Infrastrukturen von unterschiedlichen Teams realisiert werden. Eine einheitliche Ausführungsarchitektur soll dafür sorgen, daß für Business Objects, gleich wie sie realisiert wurden, immer die auf gleicher Weise über Schnittstellen miteinander kooperieren können. Darüber hinaus sollten die Eigenschaften und

Zusammenhänge der verschiedenen Business Object Typen des Metamodells festgelegt sein.

- Mehrfachverwendbarkeit: In der Ausführungsarchitektur müssen auch Schnittstellen von Business Objects zur Infrastruktur festgelegt werden. Diese Schnittstellen müssen unabhängig von einer konkreten Infrastruktur definiert sein. Ziel ist es dabei, Business Objects möglichst von der konkreten Infrastruktur-Technologie abzukoppeln, um sie einerseits in einer anderen Umgebung leichter mehrfach zu verwenden, andererseits die bestehende Technologie einfacher gegen eine neue Technologie austauschen zu können.

Am weitesten fortgeschritten sind Arbeiten zu einer solchen Ausführungsarchitektur in der Business Object Initiative (BOI) der OMG (früher Business Object Domain Task Force, BODTF). In der BOI wurde die Business Object Component Architecture (BOCA) erarbeitet [OMG98a], mit dem Ziel, eine Standard-Ausführungsarchitektur für Business Objects, in der OMG Business Object Facility genannt, zu definieren. Die BOCA umfaßt allerdings mehr als die Architektur, wie in Abbildung 3.5 dargestellt. Sie besteht aus mehreren Konzepten:

- Metamodel: Das Metamodell ist der Kern der BOCA. Es erweitert das CORBA Metamodell um Modellelemente, die Schnittstellen von Business Objects, strukturelle Beziehungen zwischen Business Objects und einfache Verhaltensspezifikationen darstellen.

- CDL: In der BOCA wird eine Definitionssprache für Business Object Spezifikationen, die Component Definition Language (CDL) vorgeschlagen. Im wesentlichen kann man mit der CDL Business Object Systeme auf Basis des BOCA Metamodells beschreiben. CDL ist eine Obermenge der CORBA IDL und erweitert sie um Sprachkonstrukte zur Beschreibung der im Metamodell erwähnten Ergänzungen.

- IDL Mapping: In der BOCA wird ein Mapping der Konzepte des Metamodells auf CORBA IDL angegeben. Damit soll eine konsistente und einheitliche Abbildung von Business Object Modellen auf eine Standard-Infrastruktur (CORBA) erreicht werden.

- Interoperability Framework: Das Interoperability Framework ist ein Satz von CORBA Schnittstellen, die ein Business Object unterstützen und nutzen muß, um technisch (im Sinne einer CORBA Infrastruktur) Interoperabel zu anderen Business Objects zu sein.

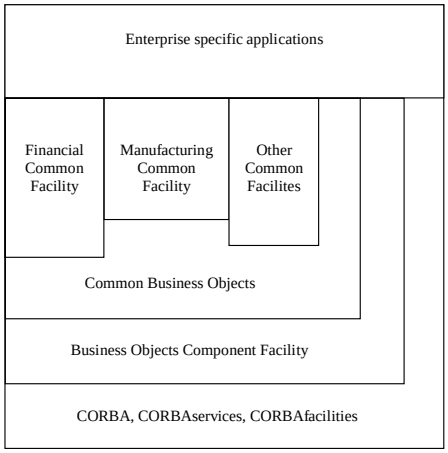

Abbildung 3.5: Business Objects Component Facility Architektur

Die BOCA wurde 1998 von den Mitgliedern der OMG als Spezifikation für eine Business Object Facility abgelehnt. Der Grund dafür ist, daß zunächst die endgültige Spezifikation für das Komponentenmodell der OMG (CORBAcomponents) [OMG98b] abgewartet werden soll.

Neben der BOCA gibt es noch weitere, allerdings proprietäre Ansätze: Wichtigster Ansatz hierbei ist das San Francisco Framework der Firma IBM [BJNR98]. San Francisco stützt sich nicht, wie BOCA auf CORBA, sondern auf Java als Ausführungsplattform. San Francisco ist bereits ein komplettes Framework mit Business Objects für verschiedene Bereiche, wie Lager-, Auftrags- oder Finanzverwaltung. Das Metamodell und die Ausführungsarchitektur von San Francisco sind im Gegensatz zu BOCA nur implizit in den Java Framework-Klassen bzw. in den Zusatzmodulen für das UML Werkzeug Rational Rose definiert. Das verhindert allerdings, daß andere Business Objects, die nicht mit San Francisco entwickelt wurden, ohne weiteres eingebunden werden können. Zur Zeit wird San Francisco auf Enterprise JavaBeans umgesetzt. Dies könnte zu einer besseren Interoperabilität mit anderen Business Objects führen.

3.3.3 Business Object Instanzarchitektur

Definition 3.6 (Business Object Instanzarchitektur) *Die Business Object Instanzarchitektur beschreibt den internen Aufbau von* `Business Objects` *auf Basis einer* `Business Object Infrastruktur`*. Änderungen der Business Object Instanzarchitektur sind transparent für die* `Komponenten`*, die ein* `Business Object` *nutzen.*

Die Business Objects Instanzarchitektur beschreibt die interne Architektur eines Business Objects. Die Instanzarchitektur ist abhängig von der Anwendungs- und Ausführungsarchitektur. Jedoch sollten Änderungen der Instanzarchitektur keine Auswirkungen auf dessen Clients haben.

In der Literatur werden verschiedene Modelle für eine Business Object Instanzarchitektur diskutiert. Wurde anfänglich noch gefordert, Business Objects sollten auch ihre Darstellung auf der GUI übernehmen [OHE96], so beschränken sich die derzeitigen Vorschläge für Business Object Instanzarchitekturen im wesentlichen auf drei Elemente:

- Interfaces bzw. Views: Business Objects werden von ihren Clients (andere Business Objects oder Elemente der Präsentationsschicht) über Schnittstellen angesprochen, die Geschäftsdaten und -funktionen des Business Objects repräsentieren. Dabei ist es sinnvoll mehrere Sichten (Views) auf ein Business Object zur Verfügung zu stellen. So „sieht“ ein Client aus dem Bereich „Personalwesen“ ein Business Object „Mitarbeiter“ anders als ein Client aus dem Bereich „Fertigungsplanung“. Allerdings sind Views nur logische Zusammenfassung von Geschäftsdaten- und Funktionen. Es sollen keine Präsentationsaufgaben, wie die Darstellung des Business Objects in der GUI damit übernommen werden.

- Factories bzw. Homes: Wenn Business Objects neu angelegt werden oder mehrere Business Objects anhand von bestimmten Kriterien gesucht werden, so muß es Schnittstellen im System geben, die von einer konkreten Business Object Instanz unabhängig sind. Hier geht es um Funktionen eines Business Object Typs. Für solche Funktionen sind Factory-Schnittstellen (nach dem Factory Pattern aus [GHJV94]) oder Home-Schnittstellen zuständig.

- Geschäftslogik: Die oben genannten Architekturelemente stellen nur Schnittstellen zur Nutzung von Business Objects dar. Die Implementierung der funktionalen Anforderungen – der Geschäftslogik – dieser Schnittstellen erfolgt in einem davon logisch getrennten Element. Abbildung 3.6 zeigt diese Instanzarchitektur schematisch. Inwieweit man eine solche Instanzarchitektur realisieren kann, hängt im wesentlichen von der verwendeten Infrastruktur ab. Im nächsten Abschnitt werden solche Infrastrukturen für Business Objects diskutiert.

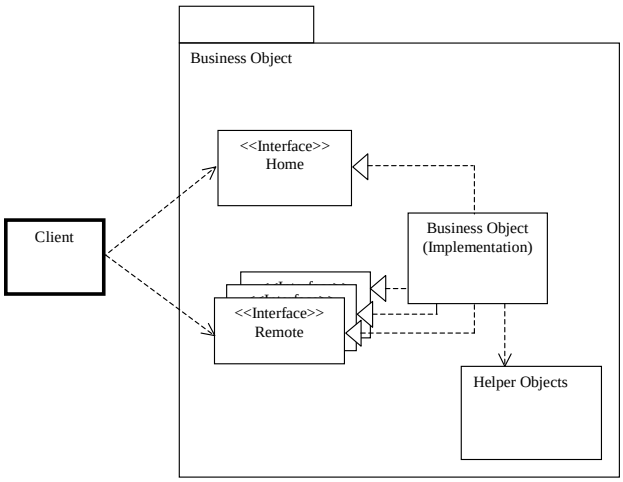

Abbildung 3.6: Business Object Instanzarchitektur

3.4 Realisierung

Business Objects setzen, wenn sie das Modell, daß sie darstellen in einem Informationssystem zur Ausführung bringen wollen, auf einer Infrastruktur (Programmiersprache, Laufzeitsystem, Transaktionsmonitor usw.) auf.

Für die Realisierung der nicht-funktionalen Anforderungen an Business Objects werden Infrastrukturen benötigt, die technische Dienste, wie Persistenz, Transaktion, Sicherheit oder Caching für die Business Objects bereitstellen, ohne daß diese direkt an diese Dienste gebunden werden. Anschaulich dargestellt wird dies durch das sogenannte „Hollywood-Prinzip“: „Don’t call us, we call you!“ [OHE96]. Nicht die Logik des Business Objects selbst kümmert sich um die Speicherung, sondern die Infrastruktur erledigt dies ohne Aufforderung. Dadurch bleibt die gleichartige Abbildung gewahrt.

Das folgende Beispiel zeigt eine Realisierung einer solchen Strategie auf Basis des Enterprise JavaBeans Frameworks. Das Business Object „Account“ wird realisiert durch ein Entity Bean `AccountBean`. Stellvertretend sind hier zwei Methoden dargestellt: `ejbStore()` für eine Methode die automatisch von der Infrastruktur – im Falle des Enterprise JavaBeans Frameworks durch den sogenannten Container – aufgerufen wird, um das `AccountBean` in der Datenbank zu speichern und `deposit()` als Methode, die Teil der Geschäftslogik des Business Objects „Account“ ist. Damit wird erreicht, daß Methoden wie `deposit()` frei von Abhängigkeiten zur Infrastruktur bleiben.

```
public class AccountBean implements EntityBean {
  private String accountId;
  private double balance;
[...]
  public void ejbStore() {
    sql = "update ACCOUNT set BAL = ? where ID = ?";
    ps = con.prepareStatement(sql);
    ps.setDouble(1, balance);
    ps.setString(2, accountId);
    ps.executeUpdate();
  }
[...]
  public double deposit(double amount) {
    balance += amount;
    return balance;
  }
}
```

Infrastrukturen die dieses Vorgehen ermöglichen werden Objekt- Transaktionsmonitore (OTM) oder Applikationsserver genannt und das Konzept an sich eine Container-Architektur.

Bekannteste Vertreter für solche Applikationsserver sind Produkte, die auf dem im obigen Beispiel bereits dargestellten von Sun entwickelten Enterprise JavaBeans (EJB) Framework [SUN00] basieren. Allerdings sind diese Produkte noch lange nicht ausgereift, es existieren wenig Erfahrungen mit ihrer Anwendung zur Erstellung großer Informationssysteme, erst recht nicht zur Erstellung von Business Object Systemen, und das EJB-Framework deckt nicht alle Anforderungen ab, die der Kerngedanke der gleichartigen Abbildung mit sich bringt.

Betrachtet man nämlich die Gestaltungselemente, die das EJB Framework bietet, um Business Objects zu realisieren, wird deutlich, daß statische Elemente der Geschäftswelt wie „Auftrag“ oder „Artikel“ mithilfe des sogenannten „Entity JavaBeans“ abgebildet werden können, es für dynamische Elemente, wie einen Vorgang oder Prozeß keine solche Entsprechung gibt. Die neben den „Entity JavaBeans“ vorhandene Variante des „Session JavaBeans“ eignet sich lediglich für sofort auszuführende, nicht über einen längeren Zeitraum dauernden Aktionen. Länger dauernde Prozesse oder gar arbeitsteilige Vorgänge werden nicht unterstützt. Diese müssen „per Hand“ in das System mit vielen Abhängigkeiten zu technischen Objekten eingebaut werden und entsprechen damit nicht den in der Einleitung dargestellten Definition von Business Objects.

3.5 Kriterienkatalog

Zusammenfassend werden in diesem Abschnitt alle Bedingungen dargestellt, die erfüllt sein müssen, damit man von einem Business Object System sprechen kann. Die Kriterien orientieren sich dabei im wesentlichen an die in den vorhergehenden Abschnitten getroffene Einteilung. Die Kriterien 1 bis 4 ergeben sich aus den Ausführungen zum Convergent Engineering und zu den Business Objects.

Kriterium 1 (Gleichartige Abbildung) *Business Objects bilden Elemente der Geschäftswelt in gleichartiger Weise in ein Informationssystem ab (siehe Definition 2.4 (Abbildung der Geschäftswelt)). In gleichartiger Weise bedeutet, daß jedes Business Object einem in sich geschlossenen Konzept der Geschäftswelt entspricht.*

Kriterium 2 (Komplette Abbildung) *Da in der Geschäftswelt sowohl informationstragende („Auftrag“, „Rechnung“ usw.) als auch prozedurale („Neuen Kunden registrieren“, „Buchung bestätigen“) Elemente eine Rolle spielen, müssen diese in gleicher Wiese im System als Business Objects abgebildet sein. Hierbei bezieht sich „komplett“ auf die verschiedenen Element-Typen, nicht auf das Modell. Ein Informationssystem wird immer nur eine Abstraktion und damit ein Ausschnitt eines realen Systems sein (siehe Definition 2.4 (Abbildung der Geschäftswelt)).*

Kriterium 3 (Abhängigkeiten) *Abhängigkeiten der Elemente der Geschäftswelt finden sich wieder in den Abhängigkeiten der Business Objects. Abhängigkeiten von Business Objects zu Objekten höherer Schichten eines Systems, wie zum Beispiel der Präsentationsschicht sind nur passiv, in Richtung des Business Objects möglich.*

Kriterium 4 (Offene Systeme) *Business Objects sind wie ihre Entsprechungen in der Geschäftswelt Elemente einer organisatorischen Einheit (z.B. einer Firma, einer Behörde oder eines Vereins), nicht aber eines Informationssystems. Das bedeutet, das Business Objects systemunabhängig und -übergreifend in gleicher Weise zugreifbar sein müssen, wie sie es systemintern sind (siehe Definition 2.18 (offenes Informationssystem)).*

Kriterium 5 (Einheitliche Modellierung) *Offene, wiederverwendbare Informationssysteme basierend auf Business Objects sind nur unter der Voraussetzung*

möglich, das sie nach einem einheitlichen Metamodell (siehe Definition 2.19 (Metamodell)) definiert wurden. Ein Referenzmodell und verschiedene Spezialisierungsstufen (siehe Abschnitt 3.2) sind darüber hinaus notwendig, wenn die Business Objects organisationsübergreifend wiederverwendet bzw. eingesetzt werden.

Kriterium 6 (Ausführungsarchitektur) *Sollen Business Objects nach dem Prinzip der gleichartigen Abbildung als Komponenten eines Informationssystems in einer standardisierten Art und Weise ablaufen und miteinander kooperieren, so muß eine Business Object Ausführungsarchitektur (siehe Definition 3.5 (Ausführungsarchitektur)) vorgegeben sein. Diese basiert auf dem Metamodell und ermöglicht Mehrfachverwendbarkeit, Interoperabilität und Systemunabhängigkeit der Business Objects.*

Kriterium 7 (Partitionierung) *Um eine Übereinstimmung mit den Abläufen und Rahmenbedingungen der Geschäftswelt zu erreichen, müssen Business Objects so partitioniert werden, daß die eigentliche Geschäftslogik von der Anwendungslogik getrennt wird. Eine Partitionierung in dieser Form erzwingt eine Infrastruktur, die nach der Container-Architektur aufgebaut ist.*

3.6 Zusammenfassung

Die Grundlage für das Metamodells zur Entwicklung unternehmensweiter System sind die Business Objects. In diesem Kapitel wurden die Arbeiten zu den Business Objects für die wesentlichen Themengebiete Modellierung, Architektur und Realisierung vorgestellt und diskutiert. Auf diese Diskussion basierend konnten dann die Kriterien definiert werden, die eine Business Object Architektur kennzeichnen.

Auf Anhand der genannten Kriterien werden im folgenden Kapitel Ansätze für eine Integration von Business Objects und Workflow-Management bewertet. Zusammen mit den im nächsten Kapitel dargestellten Anforderungen an Informationssysteme, die Workflows realisieren, bilden sie einen Rahmen für die Entwicklung des Business Object Metamodells für das methodische Framework.

Kapitel 4

Workflows

Arbeitsteilige Vorgänge (Workflows) sind wichtiger Bestandteil der Geschäftswelt. Auch sie sollten bei einem unternehmensweiten Vorgehen auf das Business Object Modell abgebildet und im Business Object System betrieben werden. Wie im vorigen Kapitel dargestellt, berücksichtigen die existierenden Ansätze zu Business Objects die Anforderungen und Besonderheiten von Workflows nicht oder nur unzureichend.

In diesem Kapitel werden zunächst die Anforderungen an eine Realisierung von Workflows dargestellt. Danach werden Standardarchitekturen für eine komponentenbasiertes Management von Workflows sowie bestehende Ansätze zu einer Integration von Informations- und Workflow-Systementwicklung vorgestellt. Ziel der Darstellungen in diesem Kapitel ist es, den Rahmen für eine Integration von Workflows in das Business Object Metamodell des methodischen Frameworks zu definieren.

4.1 Anforderungen

In diesem Abschnitt werden die Anforderungen an ein Informationssystem zusammengefaßt, die erfüllt sein müssen, um Workflows zu realisieren. Dazu ist es zunächst notwendig zu klären was unter Workflows verstanden wird. Diese Arbeit stützt sich auf die Definitionen des Arbeitskreises „Modellierung und Ausführung von Workflows“ der Gesellschaft für Informatik (GI) [JBS97].

Definition 4.1 *Workflow Ein Workflow ist eine zum Teil automatisiert ablaufende Gesamtheit von Aktivitäten, die sich auf Teile eines* `Geschäftsprozesses` *oder andere organisatorische Vorgänge beziehen. Ein Workflow besteht aus Abschnitten (Subworkflows), die weiter zerlegt werden können. Er hat einen definierten Anfang, einen organisierten Ablauf und ein definiertes Ende. Workflows sind überwiegend als ergonomische (mit Menschen als Aufgabenträgern) und nicht als technische (z.B. Einsatz von Maschinen) Prozesse zu sehen (aus [JBS97]).*

In dieser Definition werden bereits viele Charakteristika von Informationssystemen, die Workflows realisieren, genannt. Zunächst fällt solchen Systemen eine steuernde, automatisierende Rolle zu. Sie ordnen Aufgabenträgern in einem vorher festgelegten, organisiertem Ablauf Aufgaben zu, die diese in Rahmen von Aktivitäten zu bearbeiten haben. Interessant ist in diesem Zusammenhang auch die Definition eines Workflow-Management-Systems:

Definition 4.2 *Workflow-Management-System (WfMS) Ein Workflow-Management-System ist ein (re-)aktives Basissoftwaresystem zur Steuerung des Arbeitsflusses (*`Workflow`*) zwischen beteiligten Stellen nach Vorgaben eines* `Workflow-Schemas`*. Zum Betrieb eines Workflow-Management-Systems sind Workflow-Management-Anwendungen zu entwickeln. Ein Workflow-Management-System unterstützt mit seinen* `Komponenten` *sowohl die Entwicklung von Workflow-Management-Anwendungen als auch die Steuerung und Ausführung von* `Workflows` *(aus [JBS97]).*

Definition 4.3 *Workflow-Schema Ein Workflow-Schema ist eine durch ein* `Workflow-Management-System` *ausführbare Spezifikation eines* `Workflows` *als Abbildung eines* `Geschäftsprozesses` *beziehungsweise arbeitsteiligen Vorgangs. Es umfaßt die Beschreibung der Ablaufsteuerung, der Aufgabenzuweisung, der Ablauf- und Aufbauorganisation.*

Daraus ergeben sich folgende Anforderungen für Informationssysteme, die Workflows realisieren:

- Ablaufsteuerung: Die einzelnen Workflows müssen anhand einer vorher festgelegten Ablaufspezifikation (Workflow-Schema) ausgeführt werden. Die Workflows, die sich bereits in Ausführung befinden, werden Workflow-Exemplare [JBS97] genannt. Workflows müssen sich in Subworkflows gliedern lassen. Die kleinste Einheit eines Workflows ist die Aktivität.

- Aufgabenzuweisung: Das System muß automatisch anhand des Workflow-Schemas und einer Stellenbeschreibung den nächsten beteiligten Aufgabenträger ermitteln. Dabei können auch Arbeitsverteilungsstrategien berücksichtigt werden.
- Ausführung von Aktivitäten: Die Aufgaben, die in einem Workflow von einem Bearbeiter (Akteur [JBS97]) abzuarbeiten sind, werden in Rahmen von Aktivitäten (auch: Arbeitsschritten [JBS97]) erfüllt. Das System muß einen geeigneten Rahmen schaffen, damit die Werkzeuge und Daten, die zur Erledigung der Aufgaben notwendig sind, bereitstehen.

Definition 4.4 *Aktivität (Workflow) Eine Aktivität ist ein nicht weiter unterteilbarer Arbeitsschritt in einem* `Workflow`*, der von genau einem Akteur bearbeitet wird. Der Akteur wird aufgrund der Stellenbeschreibung, die der Aktivität zugewiesen ist, ausgewählt.*

Aus diesen primären Anforderungen ergeben sich weitere Eigenschaften für das Metamodell, das Form und Inhalt eines Workflow-Schemas festlegt. Dieses in [JBS97] Workflow-Metaschema bezeichnete Modell legt die Eigenschaften und möglichen Ausprägungen der verschiedenen Aspekte eines Workflow-Schemas fest. Nach [Jab95] sind folgende Aspekte relevant:

- Funktionaler Aspekt: Spezifiziert, welche Aktivitäten ausgeführt werden sollen. Hier werden logische Verarbeitungseinheiten – die Workflows – definiert. Hier spielt die Strukturierung von Workflows in Workflow, Subworkflows und Aktivitäten eine wichtige Rolle.
- Operationaler Aspekt: Beschreibt welche Hilfsmittel wie für die Erledigung einer Aufgabe verwendet werden können, wie zum Beispiel Programme oder andere Systeme.
- Verhaltensbezogener Aspekt: Definiert die Konstrukte zur Definition des Kontrollflusses innerhalb eines Workflows. Wichtige Konstrukte sind zum Beispiel die sequentielle, parallele und bedingte Ausführung von Aktivitäten eines Workflows.
- Informationsbezogener Aspekt: Beschreibt diejenigen Daten, die in den Arbeitsschritten eines Workflows konsumiert und produziert werden und in welcher Form diese zwischen den Aktivitäten ausgetauscht werden können.

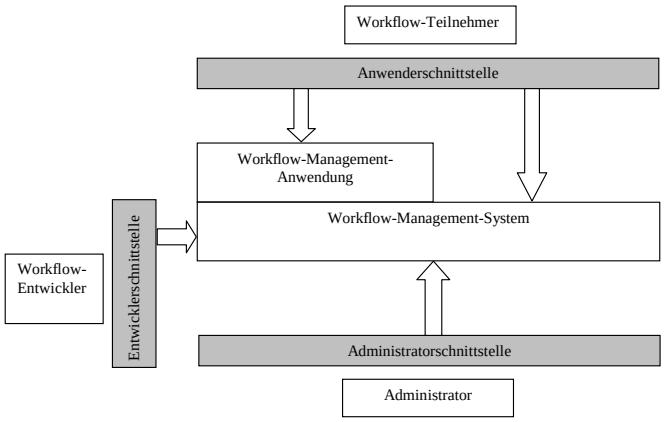

Abbildung 4.1: Wesentliche Benutzergruppen- und Schnittstellen eines WfMS

- Organisatorischer Aspekt: Spezifiziert die Elemente der Stellenbeschreibung einer Aktivität eines Workflows. Es können Merkmale festgelegt werden, mit denen automatisch ein bestimmter Bearbeiter anhand der Aufbauorganisation eines Unternehmens für die Ausführung einer Aktivität ausgewählt werden kann.

- Aufbauorganisation: Stellenbeschreibungen von Aktivitäten erfordern ein Modell der Organisation in dem ein Workflow ausgeführt wird. Dieses Modell wird in Aufbauorganisation genannt und setzt Akteure mit ihren Rollen in einer strukturierten Organisationsform in Beziehung. Die Aufbauorganisation muß auch zur Laufzeit eines Workflows zugreif- und änderbar sein, etwa im Falle einer Veränderung in der Organisationsstruktur. Hieraus resultiert nicht, daß ein Informationssystem, daß Workflows realisiert auch gleichzeitig die Aufbauorganisation einer Organisation verwalten muß, jedoch sind zumindest Schnittstellen zu den Systemen notwendig, die die Aufbauorganisation verwalten.

Die oben dargestellten Anforderungen werden üblicherweise mit Workflow-Management-Systemen (WfMS) realisiert. WfMS unterstützen die Modellierung, Ausführung und Kontrolle von längerdauernden Prozessen und arbeitsteiligen Vorgängen. Abbildung 4.1 aus [Sch99] zeigt dabei die wesentlichen Schnittstellen und Benutzergruppen eines WfMS.

Mithilfe eines Werkzeugs zur Prozeßmodellierung der Entwicklerschnittstelle werden zunächst Vorgänge definiert, d.h. wer wann welche Aufgaben innerhalb eines Prozessen ausführen soll. Diese Prozeßmodelle werden dann von der Workflow-Engine ausgeführt. Diese kümmert sich – grob dargestellt – darum,

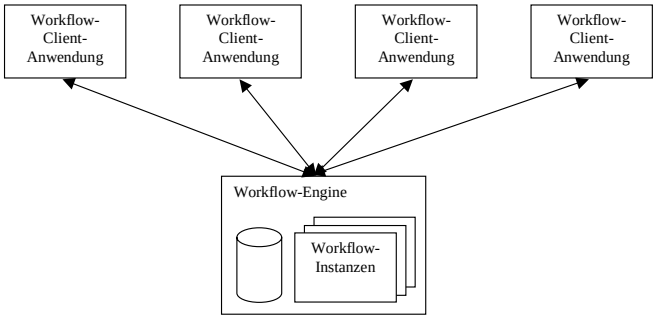

Abbildung 4.2: Client-/Server-Architektur mit zentraler Workflow-Engine

den richtigen Akteur für eine Aufgabe zu finden, ihm die Aufgabe zuzuweisen und sich nach der Ausführung zu entscheiden, welcher Schritt als nächstes auszuführen ist. Ein Akteur erhält über die Anwenderschnittstelle Informationen, welche Aufgaben er zu erledigen hat und kann diese dann über diese Schnittstelle auch ausführen, wobei Hilfsmittel und Daten, die er zur Ausführung der Aufgabe benötigt zur Verfügung gestellt bzw. aufgerufen werden. Administratoren können über die Administratorschnittstelle das WfMS konfigurieren, Fehlerzustände beheben oder analysieren.

Die Architekturen der zur Zeit auf dem Markt erhältlichen WfMS sind nahezu ausschließlich geschlossene, Client-/Server-Architekturen mit einer Workflow-Engine, wie in Abbildung 4.2 dargestellt. Das Hauptelement, die Workflow-Engine realisiert dabei den Hauptteil der oben dargestellten Anforderungen. Gleichzeitig kapselt sie die Workflow-Instanzen und verhindert so einen Zugriff anderer Systeme auf diese. Die Workflow-Clients haben nur die Aufgabe, den Benutzer einer Workflow-Anwendung die momentane Liste von Arbeitsaufträgen darzustellen, aus denen sich der Benutzer eine Aufgabe auswählen kann um sie zu starten, abzulehnen oder zu beenden. Die Workflow-Engine wird vom Workflow-Client über Statusänderungen der Arbeitsaufträge informiert.

Neben den bekannten Nachteilen einer solchen Architektur wie mangelhafte Skalierbarkeit oder Verfügbarkeit, die unter anderem in [Sch99] näher beschrieben werden, entsprechen diese Architekturen nicht den Prinzipien eines Convergent Engineering, so wie in Kapitel 3 dargestellt. Zwar werden Geschäftsprozesse und Vorgänge einer Organisation unter Umständen gleichartig in Workflow-Schemata abgebildet, jedoch sind die Workflow- Instanzen dieser Schemata nur innerhalb der Workflow-Engine sichtbar. Auch ist eine Integration mit informationstragenden Elementen der Geschäftswelt nicht gewährleistet, wenn diese in anderen Systeme realisiert wurden und keine geeignete Schnittstelle zwischen WfMS und diesen Systemen existiert.

In der Forschung und einigen Organisationen werden in letzter Zeit andere Architekturen entwickelt und vorgeschlagen. Im nächsten Kapitel werden zunächst Standard-Architekturen für Workflow-Management-Systeme vorgestellt, im nachfolgenden Kapitel neuere Ansätze zur Entwicklung objektorientierter Workflow-Systeme. Diese werden vor allem unter dem Gesichtspunkt einer möglichen Integration von Workflow und Business Objects betrachtet.

4.2 Standard-Architekturen

Workflow-Management ist aufgrund seines interdiszlipinären Charakters ein vielschichtiges Thema, das verschiedene Teilgebiete der Informatik, der Wirtschaftsinformatik und der Betriebswirtschaftslehre betrifft: Auf Basis der Analyse, Neugestaltung und Optimierung von Geschäftsprozessen, Vorgängen und Organisationsstrukturen in einem Unternehmen (Betriebswirtschaft) werden Workflow-Anwendungen mithilfe geeigneter Modellierungs- und Beschreibungstechniken erstellt (Wirtschaftsinformatik), die in der Regel von speziellen Workflow-Management-Systemen (Informatik) ausgeführt werden. In dieser Arbeit stehen vor allem die Integrations-Aspekte eines Workflow-Management-Systems im Vordergrund: Auf Basis welcher Architekturen werden diese Systeme entworfen und wie gut sind diese Architekturen in Sinne eines Convergent Engineering mit dem Konzept „Business Objects“ vereinbar?

In diesem Abschnitt werden unter diesem Aspekt die Standard-Architekturen der Workflow Management Coalition (WfMC) und der Object Management Group (OMG) vorgestellt und bewertet.

4.2.1 Workflow Management Coalition

Die WfMC ist eine 1993 gegründete Organisation, deren Mitglieder Anbieter als auch Anwender von Workflow-Produkten sind. Die Organisation hat zum Ziel, durch Schaffung von Standards die Interoperabilität von Workflow-Produkten verschiedener Hersteller zu erhöhen.

Dazu wurde 1994 ein Referenzmodell [WFM94] entwickelt, das als Basis für die Entwicklung von Integrations- und Interoperabilitätseigenschaften aller Workflow-Produkte dienen soll. Das WfMC-Referenzmodell unterteilt Workflow-Management-Systeme in mehrere Komponenten, die sich über festgelegte Schnittstellen aufrufen können. In Abbildung 4.3 ist die Referenz-

Architektur, im WfMC-Dokument als „Product Implementation Model“ bezeichnet, dargestellt.

4.2.1.1 Modell

Die WfMC-Architektur sieht vier Komponenten vor:

- die Workflow Engine, die das Workflow-Schema (Process Definition) interpretiert und ausführt, Workflow-Ausführungsinformationen (Workflow Control Data) und den Arbeitsvorrat (Work List) verwaltet, sowie Organisationsdaten (Organisation/Role Model Data) und Workflow-relevante Daten (Workflow Relevant Data) nutzt,
- das Definition Tool, mit dessen Hilfe die Workflow-Schema erzeugt und verwaltet werden,
- den Worklist Handler der den Arbeitsvorrat verwaltet und
- das User Interface mit dem die Workflow-Exemplare dargestellt und manipuliert werden können.

Die Aufteilung in diese vier Komponenten ist im Referenzmodell sehr generisch gehalten. Das Ziel der WfMC ist es, den Herstellern von Workflow-Produkten eine größtmögliche Flexibilität in der Implementierung ihrer Produkte offenzuhalten. Aus diesem Grunde stellt die WfMC-Architektur nur einen Vorschlag, jedoch kein Standard dar [WFM94]

Das eigentliche Referenzmodell besteht aus der Definition von Schnittstellen, die auf der oben genannten generischen Architektur basieren. Diese Schnittstellen sollen auf unterschiedlichen Ebenen das Zusammenspiel verschiedener Workflow-Produkte ermöglichen. Standardisiert werden sollen, wie in Abbildung 4.4 gezeigt, folgende Schnittstellen:

- zu den Prozess-Definitions-Werkzeugen (Schnittstelle 1),
- zu den Workflow-Clients, z.B. dem Worklist Handler (Schnittstelle 2),
- zu den aufzurufenden Applikationen (Schnittstelle 3),
- zu anderen Workflow-Systemen (Schnittstelle 4) und

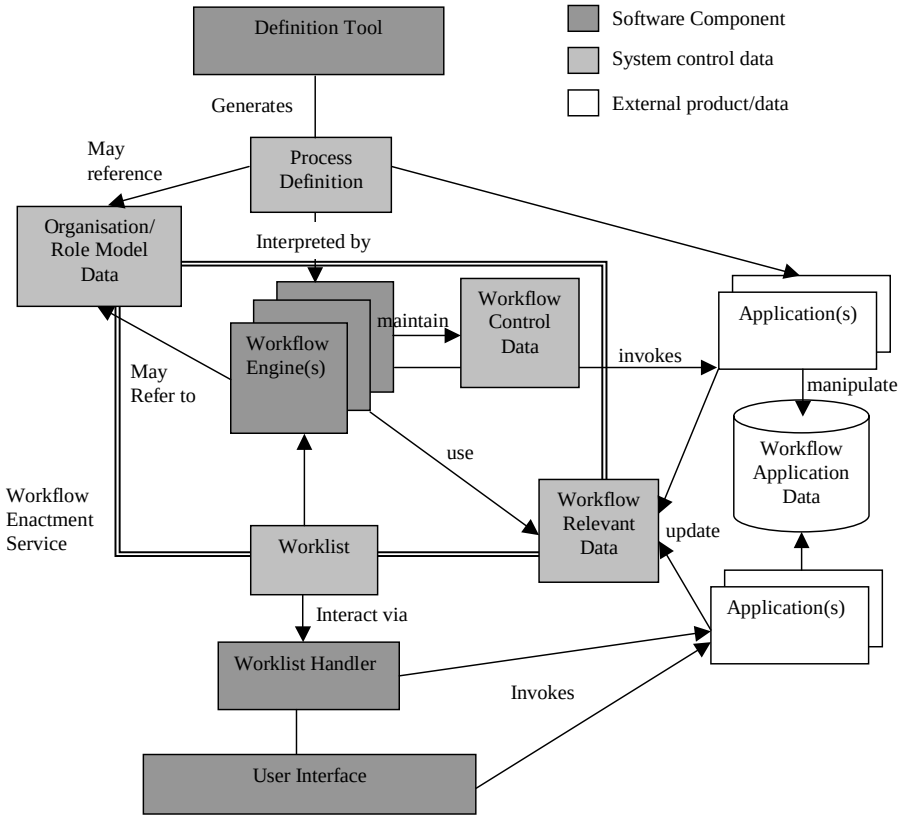

Abbildung 4.3: WfMC Product Implementation Model

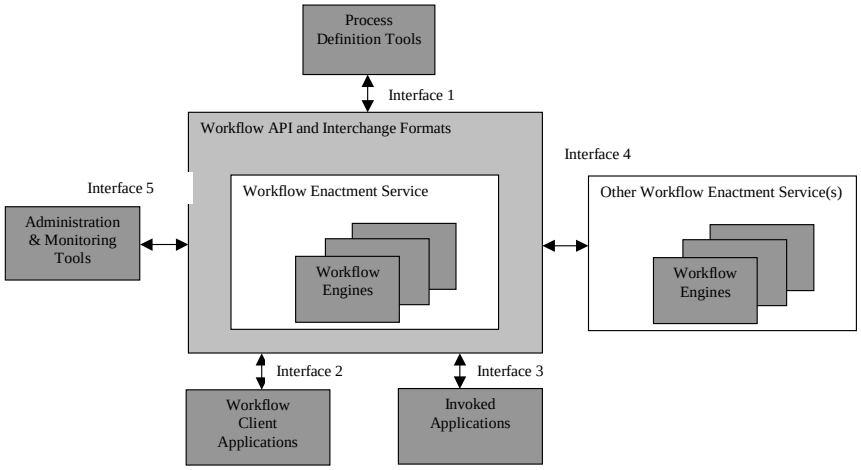

Abbildung 4.4: WfMC Referenzmodell

- zu Administrations- und Monitoring-Werkzeugen (Schnittstelle 5)

Diese Schnittstellen haben als Ausgangspunkt den Workflow Enactment Service, also einen Dienst, dessen Aufgabe es ist, mithilfe ein oder mehreren Workflow Engines Workflow- Exemplare zu erzeugen, zu verwalten und auszuführen. Die Inanspruchnahme dieses Dienstes erfolgt über eine gemeinsame Grundmenge von Funktionen der fünf oben genannten Schnittstellen, die als „Workflow Application Programming Interface and Interchange“ (WAPI) zusammengefaßt werden.

4.2.1.2 Bewertung

Das Ziel des Referenzmodells der WfMC ist es, unterschiedliche Workflow-Produkte verschiedener Hersteller miteinander zu kombinieren. Das Modell geht dabei nicht von den Workflows und Aktivitäten als elementare, zugreifbare Einheiten aus, sondern von einem zentralen Workflow Enactment Service mit entsprechenden Schnittstellen zur Erzeugung, Verwaltung und Ausführung von Workflow-Exemplaren. Damit muß, um auf die Workflow- Exemplare zugreifen zu können, die Schnittstelle einer technischen Komponente genutzt werden. Die Workflow-Exemplare können deshalb nicht als Business Objekte angesehen werden.

Aus Sicht der auszuführenden Workflows innerhalb des WfMC Referenzmodells übernimmt der Enactment Service ähnlich eines Containers alle Aufgaben zur Steuerung, Speicherung und Erzeugung von Workflow-Exemplaren. Die

Workflow-Schema sind im allgemeinen frei von Referenzen zu technischen Komponenten. Ausnahme ist die Modellierung und der explizite Aufruf von externen Anwendungen („Invoked Applications “).

Auch die Modellierung der zur (fachlichen) Steuerung des Workflows notwendigen Informationen („Workflow Relevant Data“) ist im Referenzmodell unzureichend definiert. In [WFM94] werden verschiedene Möglichkeiten diskutiert:

- Die Daten werden vom Worklist Handler verwaltet und bei Bedarf durch Kopieren an Workflow-Clients oder Anwendungen weitergegeben. Dabei soll der Worklist Handler die Daten in anwendungsspezifische Formate übersetzen.
- Die Daten werden über Referenzen auf einen gemeinsam genutzten Speicher für Ressourcen (z.B. das Dateisystem mit Dateinamen als Referenzen) weitergegeben. Wie im Abschnitt 3 dargestellt, sollten im Sinne eines Convergent Engineerings die notwendigen Informationen für Workflow Business Objects Referenzen auf andere Business Objects sein. Dementsprechend ist die zweite Möglichkeit vorzuziehen. Allerdings muß hier seitens der WfMC angegeben werden, wie solche Referenzen in das Referenzmodell integriert werden sollen.

Da das Referenzmodell der WfMC zu generisch ist, kann seine Eignung für eine Integration von Business Objects und Workflows nur anhand der Schnittstellendefinitionen des WfMC Referenzmodells getroffen werden. Demnach ist keines der Kriterien aus 3.5 erfüllt. Allerdings war eine Integration mit dem Business Objects Konzept auch nie Ziel der WfMC. Das die Vorstellung der WfMC-Konzepte dennoch sinnvoll ist, zeigt sich im nächsten Abschnitt, in dem die Workflow-Management-Facility der OMG vorgestellt wird, die auf die Arbeiten der WfMC aufbaut.

4.2.2 Object Management Group

Ebenso wie die WfMC ist die Object Management Group (OMG) eine unabhängige Interessengemeinschaft von Software-Herstellern und Anwendern. Die OMG existiert seit 1989 und hat sich zum Ziel gesetzt, ein umfassendes Referenz-Architekturmodell für objektorientierte, verteilte Anwendungen zu definieren [OMG92]. Hierbei sollen nicht nur standardisierte Spezifikationen von Schnittstellen und Komponenten des Architekturmodells erarbeitet werden, sondern

auch Spezifikationen der fachlichen Modelle verschiedener Anwendungsbereiche, wie zum Beispiel der Medizin-, Produktions- oder Telekommunikationsbranche [OMG95].

Das grundsätzliche Architekturmodell und integrierendes Moment der OMG Spezifikationen ist die Object Management Architecture (OMA). Die OMA ist ein Architekturmodell, das verteilte, miteinander durch Methodenaufrufe kommunizierende Objekte als Ausgangspunkt für die Entwicklung von Anwendungssystemen vorsieht. Die Schnittstellen dieser Objekte werden durch eine sprach- und plattformunabhängige Schnittstellen-Beschreibungssprache, der Interface Description Language (IDL) definiert. Dadurch können Objekte miteinander verbunden werden, die in unterschiedlichen Programmiersprachen für verschiedene Plattformen geschrieben wurden. Die Kommunikation der Objekte untereinander erfolgt dabei über den Object Request Broker (ORB). Ein ORB ist eine logische Softwarekomponente, die eine Infrastruktur für die Kommunikation verteilter Objekte bereitstellt.

Die OMG selber stellt keine ORBs zur Verfügung, sondern definiert nur die plattform- und sprachunabhängigen Schnittstellen, Austauschformate und das Verhalten eines ORBs. Dieser Standard wird Common Request Broker Architecture (CORBA) [OMG92] genannt und umfaßt das CORBA Objektmodell, die Schnittstellen-Beschreibungssprache IDL, die Spezifikation des eigentlichen ORBs und Festlegungen zur Interoperabilität zwischen verschiedenen ORBs. Neben CORBA umfaßt die OMA die Spezifikationen für CORBAservices, CORBAfacilities und Domain Interfaces. Bei den CORBAservices handelt es um grundlegende Dienste, die in verteilten, objektorientierten Systemen benötigt werden. Beispiele sind der Naming Service zur Zuordnung und Adressierung von Objekten innerhalb eines logischen Namensraumens oder der Transaction Service zur Abwicklung transaktionaler Methodenaufrufe. Die Domain Interfaces sind Schnittstellen für fachliche Objektmodelle der oben erwähnten Anwendungsbereiche.

Die für die in dieser Arbeit geführte Diskussion einer Verbindung von Business Objects und Workflow-Management wichtige Spezifikation einer Workflow Management Facility ist Teil der CORBAfacilities. CORBAfacilities sind Schnittstellendefinitionen für nicht-fachliche Anwendungsdienste. Dazu gehören Schnittstellen zur Nutzung einer plattformunabhängigen Benutzerschnittstelle (User Interface Common Facilities), zur Nutzung administrativer Dienste zur Verwaltung verteilter Systeme (System Management Common Facilities), zur Nutzung von Diensten zur Speicherung, zum Austausch und Verwaltung von Dokumenten (Information Management Common Facilities) und zur Nutzung von Diensten zur

Ausführung und Verwaltung von Arbeitsabläufen (Task Management Common Facilities), denen auch die Workflow Management Facility zugeordnet ist. Zu ihnen gehört auch die im Abschnitt 3.3.2 vorgestellte Business Objects Component Architecture (BOCA).

4.2.2.1 Anforderungen

Neue CORBAfacilities werden nicht von der OMG selbst ausgearbeitet, sondern es wird zunächst ein Request for Proposal (RFP) ausgeschrieben, in denen die Anforderungen an die zu definierende Facility gestellt werden. Daraufhin können Mitglieder oder eine Gruppe von Mitgliedern der OMG Vorschläge einreichen. Über diese Vorschläge wird dann von allen OMG-Mitgliedern abgestimmt, ob der jeweilige Vorschlag angenommen oder abgelehnt wird. Jedoch wird ein angenommener Vorschlag nur dann zu einer öffentlichen OMG Definition, wenn innerhalb eines Jahres nach Annahme ein „Proof-of-Concept“, daß heißt die Realisierung des Vorschlages durch OMG Mitglieder, erfolgt ist. Der RFP für die Workflow-Management-Facility [OMG97a] wurde im Mai 1997 veröffentlicht. Im Folgenden werden die Anforderungen des RFPs kurz skizziert, da sie zeigen, daß an sich eine Integration von Workflow und Business Objects ein Teilziel des RFPs war.

Der RFP gliedert sich in notwendige und optionale Anforderungen. Notwendige Anforderungen umfassen Punkte, die in einem Vorschlag berücksichtigt werden müssen, wohingegen optionale Anforderungen von den Einreichern von Vorschlägen behandelt werden können.

Notwendige Anforderungen des RFP betreffen Durchführung und Monitoring von Workflows einschließlich der Integration externer Applikationen und Objekte. Zudem sollen

- das Workflow-Metaschema des Vorschlages beschrieben werden,
- die Schnittstellen zur Durchführung, Monitoring und Verfolgung (History) von Workflows spezifiziert werden,
- verschachtelte Workflows unterstützt werden.

Die optionalen Anforderungen beschäftigen sich in erster Linie mit der Definition (Erstellung von Workflow-Schemata) von Workflows. Es können

- Schnittstellen zur Definition und Manipulation von Workflow-Schemata spezifiziert,

- Verschachtelte Workflow-Schemata zugelassen und
- Möglichkeiten zur Unterstützung sogenannter ad-hoc Workflows angegeben werden.

Darüber hinaus gibt es Standardanforderungen an alle Einreichungen für Elemente der OMA. Für diese Arbeit wichtig ist unter anderem die Forderung, bestehende Services und Facilities der OMA wiederzuverwenden und auf ihnen aufzubauen. Eines der wichtigsten Ziele der OMA ist die Verwirklichung des Bauhaus-Prinzips [OMG92], das heißt, das ein Service oder eine Facility von allen Objekten der OMG verwendet werden kann, aber möglichst nur einmal und unabhängig von anderen definiert und implementiert werden darf. Deshalb ist für diese Arbeit wichtig zu beurteilen, inwieweit die Workflow-Management- Facility mit der Business Object Facility zusammenarbeitet.

Auf den RFP wurden vier Vorschläge eingereicht, angenommen wurde der jointFlow- Vorschlag, der von der WfMC und anderen Mitgliedern gemeinsam eingereicht wurde. Das jointFlow Modell wird im Folgendem vorgestellt und bewertet.

4.2.2.2 Modell

Die jointFlow Workflow-Management-Facility lehnt sich konzeptionell sehr stark an das im letzten Abschnitt vorgestellte WAPI Schnittstellenmodell der WfMC an. Abbildung 4.5 zeigt einen Überblick über das jointFlow zu Grunde liegende Workflow-Metaschema. Zentraler Bestandteil ist das WfProcessMgr Interface. Es bietet folgende Funktionalität:

- Zugriff auf die im WfMS vorhandenen Prozeßinstanzen, Aktivitäten und Workitems (Gerade ausgeführte Aktivitäten), in dem es Referenzen auf WfProcess Interfaces liefert.
- Erzeugung einer neuen Prozessinstanz mit der Funktion create_process()
- Zugriff auf Meta-Informationen über den Kontext, den ein Prozeß benötigt und die Resultate die ein Prozeß erzeugt hat.

WfProcessMgr spielen in jointFlow die Rolle eines Templates für eine konkrete Workflow- Instanz und werden in jointFlow als per-se existent betrachtet. Das

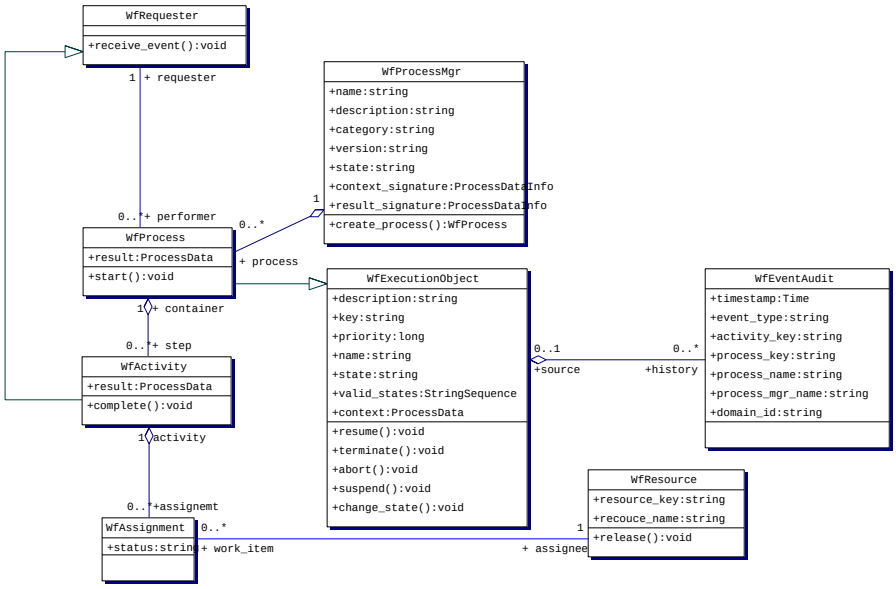

Abbildung 4.5: Joint Workflow Management Facility Model

heißt, es gibt keine Factories, die sie erzeugen. Objektreferenzen auf WfProcessMgr können z.B. über einen Naming Service bezogen werden. Es gibt jedoch keine explizite Schnittstelle für Workflow-Schemata.

Das Interface WfProcess ist ebenso wie das Interfaces WfActivity eine Spezialisierung von WfExecutionElement (und damit auch die hiervon abgeleiteten Interfaces). Beide bieten folgende Funktionalität:

- Ermitteln des augenblicklichen Zustandes des Elements.
- Ermitteln der vom augenblicklichen Zustand aus erreichbaren Zustände.
- Ermitteln der vom augenblicklichen Zustand aus möglichen Transitionen.
- Ausführen von Zustandsübergängen.
- Ermitteln bzw. Ändern der mit dem Element assoziierten Person.
- Zugriff auf History-Informationen des Elements über das WfEventAudit Interface.

In Abbildung 4.6 wird das Zustandsübergangsdiagramm eines WfExecutionObjects dargestellt.

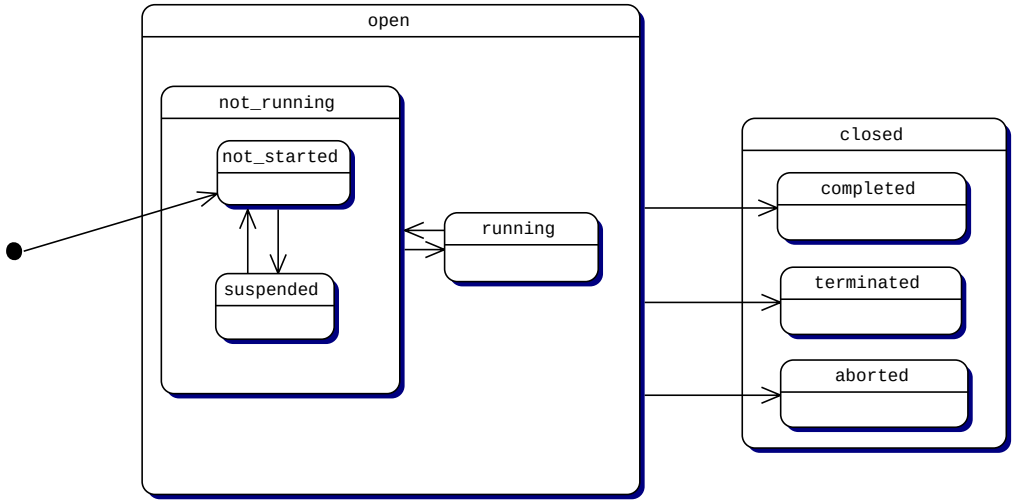

Abbildung 4.6: Zustandsdiagramm eines WfExecutionObjects

WfProcess Objekte repräsentieren eine Workflow-Instanz und werden von WfProcessMgr Objekten erzeugt. Ein WfProcess umfaßt im Allgemeinen mehrere WfActivity Objekte. WfActivity Objekte sind vom WfRequester Interface abgeleitet und können ihrerseits wieder WfProcess Objekte starten. Auf diese Weise werden verschachtelte Workflows möglich.

Eine WfActivity beschreibt eine Aktivität innerhalb eines Workflows. Einer WfActivity sind ein oder mehrere WfAssignments zugeordnet. WfAssignments binden Bearbeiter (WfResources) an eine Aktivität. In jointFlow können also mehrere Bearbeiter in einer Aktivität involviert sein. Ein WfAssignment Objekt stellt die Verbindung eines mit genau einem WfResource (Bearbeiter) Objekt her. WfResource Objekte dienen in jointFlow lediglich als Referenz auf bereits vorhandene oder im Rahmen der Business Objects Facility noch zu definierende Objekte.

4.2.2.3 Bewertung

Zwar werden von jointFlow alle notwendigen sowie einige der optionalen Anforderungen erfüllt, jedoch wird das Modell der Forderung, den verteilten, objektorientierten Charakter der OMA zu berücksichtigen, nicht gerecht.

Das würde nämlich bedeuten, daß eine Workflow-Management-Facility ihre Funktionalität – also in erster Linie die Fähigkeit, die Durchführung von Workflows zu steuern – aus dem Zusammenspiel autonomer, aktiver Objekte bezieht.

Ein entscheidender Grundgedanke hinter der OMA ist ja die Möglichkeit, Systeme aus einer Fülle aktiver und zu einem gewissen Grad autarker Instanzen zusammen zu setzen.

Das trifft jedoch für jointFlow nicht zu. Vielmehr wurde, wie auch beim WAPI Object Binding der WfMC, offensichtlich eine objektorientierte Schnittstelle zu monolithischen WfMS geschaffen. Die Frage danach, wie die Durchführung eines Workflows realisiert wird, ist nicht beantwortet worden. Ein Beispiel für dieses Defizit: Wenn eine WfActivity beendet ist, d.h. in einen der Unterzustände von „closed“ wechselt, so wird es im Normalfall notwendig sein, eine weitere Activity zu starten. Dies wird vom Ergebnis abhängen, mit dem die Activity beendet wurde. Es ist jedoch aus dem jointFlow Workflow-Metaschema jedoch nicht ersichtlich, welche Instanz diese Entscheidung treffen und danach die neue Activity starten soll. Hier verläßt man sich scheinbar auf einen im Hintergrund laufenden Workflow Enactment Service, der wie in der monolithischen WfMC Architektur alleine für die Abarbeitung der Workflow-Instanzen verantwortlich ist [Sch97].

Der aus der Sicht einer Integration von Workflow-Management und Business Objects wichtigste Kritikpunkt ist die mangelnde Bezugnahme auf eine OMG Business Objects Facility. Dabei liegt der Grund für dieses Defizit nicht direkt am der jointFlow-Architektur, sondern an dem schon erwähnten Bauhaus-Prinzip. Workflows sind im Sinne eines Convergent Engineering Business Objects. In Kriterium 2 (Komplette Abbildung) wird gefordert, daß nicht nur informationstragende sondern auch prozedurale Elemente der Geschäftswelt als Business Objects abgebildet werden müssen. Das bedingt die in Kriterium 5 (Einheitliche Modellierung) gemeinsame Modellierung von Workflows in einem Business Objects Metamodell. Dies ist bei jointFlow nicht gegeben. Zwar können Workflow-Anwendungen, die auf jointFlow basieren offener gestaltet werden, eine gleichartige Abbildung von Business Objects ist damit aber nicht gegeben.

4.3 Ansätze zur Integration

Im folgenden Abschnitt werden Ansätze vorgestellt, die explizit eine Integration von Business Objects und Workflow-Management anstreben. Allerdings können zu solchen Ansätzen zur Zeit nur drei Vorschläge gezählt werden. Es gibt inzwischen viele Vorschläge, die verteilte und objektorientierte Architekturen für Workflows einführen, jedoch ohne diese mit Business Objects Architekturen zu koppeln, bzw. Workflows als Business Objects zu definieren. Zu diesen Arbeiten zählen:

- WASA: Im Projekt WASA [WVBMP98] wird eine verteilte Architektur für Workflow-Systeme auf Basis von CORBA vorgestellt. Schwerpunkt des Projektes sind jedoch Fragen zur Modellierung von Workflows zur Unterstützung naturwissenschaftlicher Experimente.

- METEOR^2: In diesem Projekt [MSKW96] wurden verschiedene Architekturen für verteilte Workflow-Systeme entwickelt. In der ORBwork [KS97] Architektur wird CORBA als Grundlage verwendet. Hauptziel des Projektes ist aber die Untersuchung der Zuverlässigkeit von Workflow-Systemen.

- BPAframe: Mithilfe von CORBA wurde hier ein Workflow-Management-Framework für agentenbasiertes Workflow-Management [Mit97] entwickelt. Aus diesem Grund wurde für das Architekturmodell auch ein stark dezentralisierter Ansatz gewählt. Business Objects spielen in diesem Projekt jedoch keine Rolle.

- MicroWorkflow: Ziel dieses Projekts [MJ99] war es, ein objektorientiertes Basisframework mit den grundlegendsten Bausteinen für die Realisierung von Workflows bereitzustellen. Hauptziel war es zu zeigen, das mithilfe eines solchen Frameworks komplexe Workflow-Systeme erstellt werden können. Obwohl MicroWorkflow eine verteilte, objektorientierte Architektur besitzt, spielte Convergent Engineering bei der Konzeption keine Rolle.

4.3.1 WorCOS

Ein Ziel des Projektes WorCOS und des in [Sch99] vorgestellten Modells war es unter anderem „das Verhältnis von Workflow-Management und Geschäftsobjekten [zu] diskutieren“ und es sollte „ein Lösungsweg für eine Integration aufgezeigt werden“ [Sch99]. Hauptziel des Projektes war die Konzeption und Entwicklung einer dienstbasierten Workflow- Management-Architektur auf Basis von CORBA. Entwickler von Workflow-Management- Anwendungen sollen die WorCOS-Basisdienste für die Verwaltung von Workflow-Schemata und Workflow-Instanzen wie eine Middleware in Anspruch nehmen.

Das Ziel einer Integration wurde jedoch nicht erreicht. Business Objects spielen nur eine Rolle als Teil eines Anwendungs- und Ressourcen-Schemas für den operationalen Aspekt (siehe 4.1). Auch ist nicht klar, ob die in WorCOS ausgeführten Workflow-Exemplare selbst Business Objects sind.

Die Architektur ist an sich offen: Man kann Workflows, die in WorCOS aus Composite und Elementary Workflows zusammengesetzt sind über CORBA-

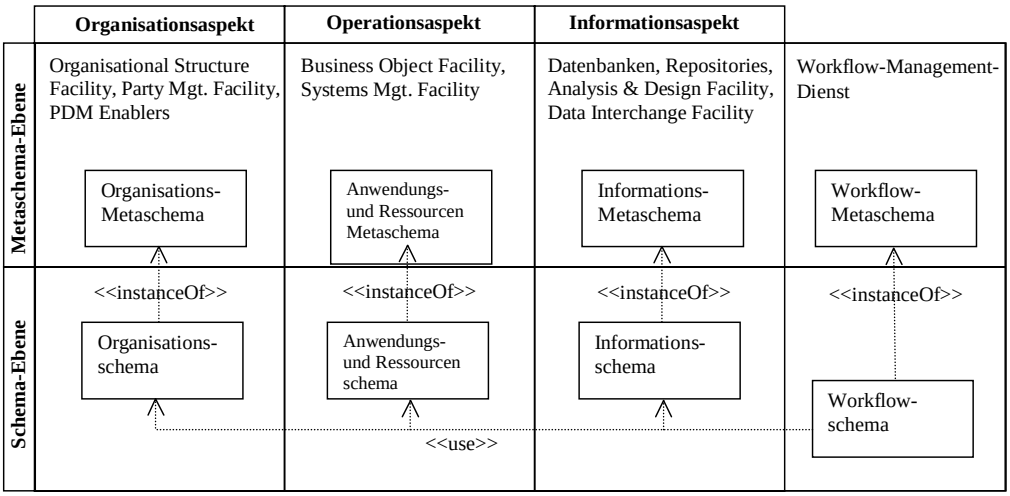

Abbildung 4.7: Beziehung des Workflow-Metasch. zu anderen Metasch. (aus [Sch99])

Referenzen ansprechen. Auch die Steuerung erfolgt nicht durch einen monolithischen Workflow-Enactment-Service, sondern wird von den Workflow-Objekten selbst durchgeführt. Damit sind die in Abschnitt 3.5 genannten Kriterien 1 (Gleichartige Abbildung), 2 (Komplette Abbildung) und 4 (Offene Systeme) erfüllt.

Die Workflow-Objekte in WorCOS sind von der Konzeption her allerdings eigenständige, aktive CORBA-Objekte, das heißt: Sie rufen von sich aus die Dienste von CORBAservices und CORBAfacilities auf. Dadurch entstehen Abhängigkeiten zu Persistenzmechanismen, Anwendungen oder Präsentations-Elementen, wie einer Worklist. Das bedeutet das die Kriterien 3 (Abhängigkeiten) und 7 (Partitionierung) nicht erfüllt sind. Hier wäre eine Container-basierte Architektur besser gewesen.

Es wird nicht deutlich, ob Kriterium 5 (Einheitliche Modellierung) erfüllt ist. Einerseits erscheinen Referenzen zu Business Objects im WorCOS Workflow-Metaschema, jedoch ist aus Abbildung 4.7 (aus [Sch99]) ersichtlich, daß Workflow-Metaschema und Anwendungs- und Ressourcen-Schema, das wie oben dargestellt zum Teil auf dem Business Object Metamodell der Business Object Facility aufbaut, vorhanden aber nicht integriert sind. Aufgrund des Fehlens eines gemeinsamen Metamodells kann auch Kriterium 6 (Ausführungsarchitektur) nicht erfüllt sein, da eine solche Architektur auf einem gemeinsamen Metamodell basieren muß.

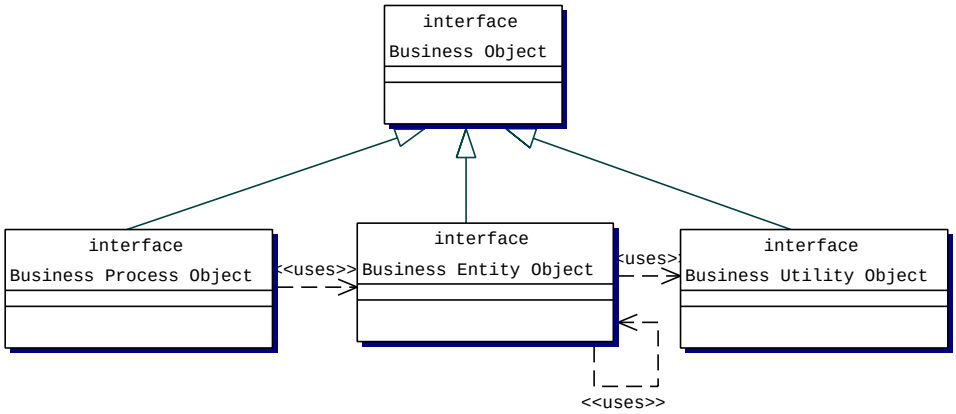

Abbildung 4.8: BOF Lite Metamodell (Ausschnitt)

Im Vergleich zu den im letzten Kapiteln vorgestellten Standard-Architekturen ist WorCOS weitreichender, obwohl die angestrebte Integration von Workflow-Management und Business Objects nur ansatzweise erkennbar ist.

4.3.2 BOF Lite

Die Business Object Facility (BOF) Lite, vorgestellt in [ES98] ist eine vereinfachte Implementierung des BOCA Vorschlags an die OMG (siehe 3.3.2). Im Metamodell von BOF Lite sind Business Entity Object und Business Process Object gleichberechtigte Business Objects, wie in Abbildung 4.8 dargestellt.

BOF Lite erfüllt die Kriterien 3-7 vollständig. Jedoch wird BOF Lite den Anforderungen an Workflow-Management Systeme wie in Abschnitt 4.1 dargestellt nicht gerecht. Unter Business Process Objects werden nämlich nur solche Prozesse verstanden, die von einem Benutzer bearbeitet werden. Grundlegendes Kennzeichen von Workflows ist aber gerade die Durchführung eines arbeitsteiligen Vorgangs. Damit können wesentliche Teile eines Geschäftsmodells nicht auf das BOF Lite Modell abgebildet werden, weshalb die Kriterien 1 und 2 nicht erfüllt werden.

Leider wurde Das Projekt BOF Lite nicht weitergeführt. Die Ergänzung um Business Workflow Objects und die Einbindung von Workflow-Diensten in die Facility wäre eine erster Beweis für die Machbarkeit einer Integration von Workflow und Business Objects gewesen.

4.3.3 BOMA

Die Business Object Management Architecture (BOMA) [Mar00] ist eine Modellierungskonzept für Geschäftsmodelle und deren Abbildung in Informationssysteme. Die Modellierung erfolgt auf Basis von vier Modellelementen:

- Purpose: Mit diesen Modellelementen werden Komponenten der Unternehmung und des Systems definiert, die Bestimmen, welche Ziele im Unternehmen zu erreichen sind und welche Ergebnisse dabei erzielt werden. Im Informationssystem übernehmen diese Komponenten eine messende bzw. überwachende Rolle gegenüber den Prozessen ein.
- Process: Prozesse definieren, wie eine Organisation ihre Ziele erreichen können. Dabei sollen Prozesse einen wertsteigernd für das Unternehmen sein. Sie sind hierarchisch aus „unteilbaren“ Prozessschritten aufgebaut, die untereinander durch Workflow-Regeln miteinander verbunden. Ein Prozessschritt ist einer organisatorischen Rolle zugeordnet um einen arbeitsteiligen Prozess zu ermöglichen. Abbildung 4.9 zeigt, wie Prozesse mit anderen Modellelementen zusammenhängen.
- Entity: Diese informationstragenden Elemente werden erzeugt, benutzt und gelöscht von Prozessen und modelliert über die Rollen, die sie in einem Prozess einnehmen. Eine Rolle hängt vom Kontext ab, in welcher das Entity benutzt wird, jedoch ist sind die Rollen eines Entities wiederverwendbar zwischen Prozessen und sein Zustand wiederverwendbar von anderen Entities und Organisationen.
- Organisation: Diese Modellelemente bilden die Aufbauorganisation und Rollen eines Unternehmens ab.

Diese vier Modellelemente bilden das Business Object Metamodell der BOMA. Für die BOMA gibt es eine prototypische Referenzimplementierung in Java [Mar00]. Workflows sind in BOMA mithilfe von Java-Klassen definiert, die von der in Abbildung 15 dargestellten Generic Process Klasse abgeleitet sind. Diese Klasse implementiert die generischen Workflow-Funktionen: Neue Prozesse erzeugen, Zuordnung von geeigneten Rollen, Verwaltung von Status und Fortschritt der Prozesse. BOMA realisiert diese Konzepte in einem Java-Framework in der Form, daß ein Programmierer die nur Business Logik implementieren muß, der Rest wird vom System generiert.

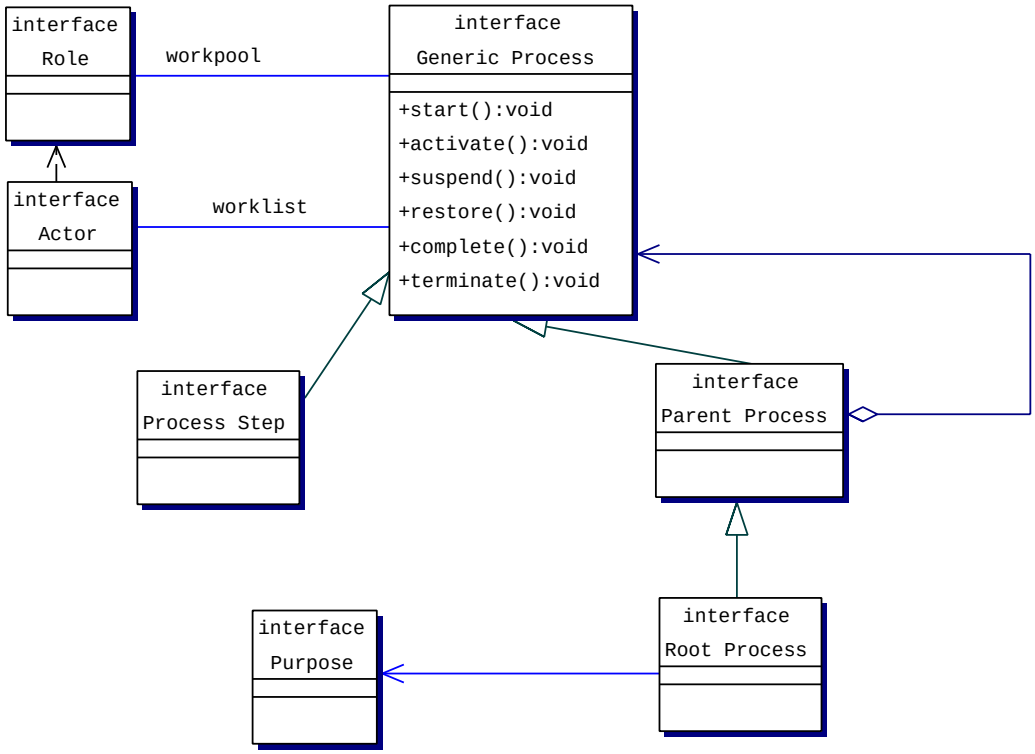

Abbildung 4.9: BOMA Process Modell-Element

Die BOMA erfüllt nahezu alle Anforderungen und Kriterien, die in dieser Arbeit vorgestellt werden. Einzig Kriterium 4 (Offene Systeme) wird nicht erfüllt, da BOMA ähnlich wie das IBM San Francisco Framework (siehe Abschnitt 3.3.2) direkt auf Java aufsetzt und eine Interoperabilität mit anderen Systemen dadurch nicht im Sinne eines Convergent Engineering möglich ist.

4.4 Zusammenfassung

In diesem Kapitel wurden Anforderungen an eine Realisierung von Workflows bei der Entwicklung von Informationssystemen dargestellt. Die primären Anforderungen sind Aufgabenzuweisung und Ablaufsteuerung. Aus diesen ergeben sich verschiedene Aspekte, die bei der Realisierung von Workflows berücksichtigt werden müssen.

Für Workflows auf Basis von Business Objects sind komponentenbasierte Architekturen für das Workflow-Management notwendig. Daher wurden in diesem Kapitel Standardarchitekturen in diesem Bereich und bestehende Ansätze zur Integration von Informations- und Workflow-Systementwicklung vorgestellt.

Diese Anforderungen und Aspekte gelten auch für die Abbildung von arbeitsteiligen Vorgängen auf ein Business Object System. Die bestehenden Ansätze und Architekturen dienen dabei als Grundlage für die Realisierung einer solchen Abbildung. Beides definiert den Rahmen für eine Integration von Workflows in das Business Object Metamodell.

Zu Beginn des nächsten Teils der Arbeit wird das Business Object Metamodell als Basis des methodische Kapitels entwickelt. Im nächsten Kapitel werden dafür die Entwurfsgrundsätze für den Aufbau und die Struktur des Metamodells festgelegt.

Kapitel 5

Entwurfsgrundsätze

Bevor im nächsten Kapitel 6 schrittweise das Business Object Metamodell als Kern des in dieser Arbeit vorgestellten Frameworks entwickelt wird, werden in diesem Kapitel die Entwurfsgrundsätze für dieses Metamodell diskutiert. Zunächst wird festgelegt, welche Sichten und Elemente aus den Bereichen Unternehmensmodellierung, Systementwicklung und Infrastruktur im Modell berücksichtigt werden sollen. Danach werden grundlegende Designentscheidungen vorgestellt, die funktionale wie nicht-funktionale Eigenschaften des Modells betreffen (siehe Definitionen 3.1 (funktionale Anforderungen) und Definition 3.2 (nicht-funktionale Anforderungen)). Eine nicht-funktionale Eigenschaft ist die Verwendung der Managed Object Facility der OMG als Metamodells des Business Object Metamodells. Da diese Entscheidung die Struktur des Metamodells stark beeinflußt, werden die wichtigsten Merkmale des MOF Metamodells am Schluß des Kapitels genauer diskutiert.

5.1 Gemeinsames Modell

Hauptziel des in dieser Arbeit vorgestellten integrierten Business Object Metamodells ist es, Teile der Unternehmensmodellierung, der konventionellen Systementwicklung und der Infrastruktursicht in ein gemeinsames Modell einzubeziehen. Wie im Abschnitt 1.2 bereits beschrieben, werden in diesen drei Bereichen teils eigene Modelle und Modellkonzepte entwickelt, teils werden gemeinsame Elemente verwendet, jedoch oftmals auf unterschiedliche Art und Weise.

Bei der klassischen Unternehmensmodellierung wird vorrangig eine Unterneh-

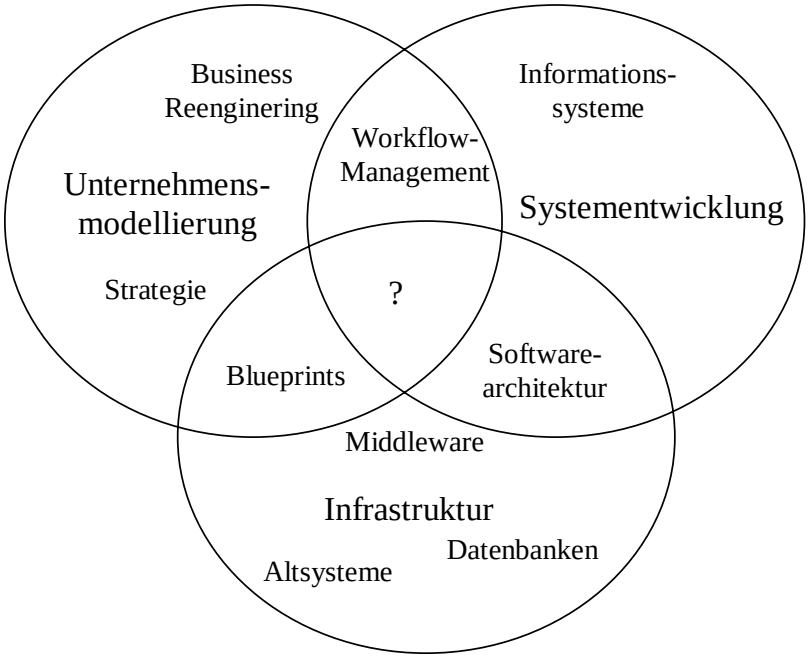

Abbildung 5.1: Gesucht: Ein gemeinsames Modell aus unterschiedlichen Bereichen

mensarchitektur analysiert oder entwickelt. Hier treten vor allem die wirtschaftlichen Aspekte in den Vordergrund: Was ist der Sinn und Zweck einer Unternehmung, eines Prozesses oder einer Aktivität? Welchen Nutzen oder Profit hat das Unternehmen von bestimmten Geschäftsprozessen und wie kann man ihn erhöhen? Welche Organisationsform ist geeignet, um bestimmte Tätigkeiten effektiv durchzuführen? Neuere Formen der Unternehmensmodellierung versuchen, vor allem durch eine Abbildung von Geschäftsprozessen auf Workflowmanagement-Anwendungen, Systementwicklung und Unternehmensmodellierung zu verbinden [SRS96]. Auf der anderen Seite wird zunehmend die Bedeutung der Soft- und Hardwarelandschaft eines Unternehmens für dessen Erfolg erkannt. Sogenannte Blueprints [IBM93] beschreiben die bestehende Abbildung von Geschäftsprozessen auf Soft- und Hardware einerseits und schränken andererseits die Entwicklung neuer Systeme mit Vorgaben für die zu verwendende IT-Infrastruktur ein.

Die konventionelle Systementwicklung beschäftigt sich vorrangig mit der Erstellung *eines* Systems innerhalb *eines* Projektes. Kennzeichen hierfür ist, daß der Ausgangspunkt der meisten heutigen Softwareentwicklungsmethoden Use Cases (Nutzungsfälle) sind (z.B. Catalysis, Rational Unified Process oder SELECT [DW99, Kru98, AF98]). Use Cases beschreiben, welche Akteure in welcher Rolle Dienste eines Systems in Anspruch nehmen und was die Aufgabe dieser Dienste

ist. Eine entscheidene Rolle spielt hierbei die Systemgrenze. Außerhalb dieser Grenze werden die Akteure dargestellt, innerhalb die Dienste. Das heißt, daß das zu entwickelnde System von Anfang an isoliert von Diensten anderer Systeme gesehen wird. Systemübergreifende oder gar unternehmensweite Zusammenhänge treten dabei in den Hintergrund, was dazu führt, das Softwaresysteme entstehen, die Geschäftsprozesse redundant und auf unterschiedliche Art und Weise implementieren. So wird in einem großen Unternehmen im Laufe der Zeit die Softwarelandschaft zunehmend heterogener und es wird schwierig, Systeme zu integrieren oder an sich verändernde Geschäftsprozesse anzupassen.

Die Infrastruktursicht ist vor allem durch die verwendeten Produkte und bestehenden Systeme geprägt. Ein nicht unerheblicher Teil der Entwicklungszeit wird in die Anpassung eines in der Analyse und frühen Designphase entwickelten fachlichen Modells an die gewählte Infrastruktur verwendet. Hier gilt es, fachliche Konzepte mit technischen (Hilfs-)Mitteln zu verbinden. Seit Mitte der 90er Jahre wird mithilfe der sogenannten Middleware versucht, immer wiederkehrende Probleme einer solchen Anpassung zu vermeiden, in dem man Bibliotheken, Frameworks oder Komponenten verwendet, die auf einer hohen Abstraktionsebene Dienste wie Verteilung, Persistenz, Transaktionskontrolle, Authentifizierung oder Autorisierung zur Verfügung stellen. Die in Abschnitt 3.3 beschriebene Object Management Architecture (OMA) der OMG ist ein Beispiel für den zur Zeit in dieser Richtung am weitgehendsten Ansatz. Doch trotz dieser Entwicklung bleibt der Nachteil, ein fachliches Modell durch die Elemente der Softwarearchitektur eines Systems zu ergänzen und damit an ein System zu binden.

Das in dieser Arbeit entwickelte Modell muß Konzepte aus Teilen der verschiedenen Sichten enthalten. Dabei ist es jedoch nicht notwendig alle Aspekte der drei genannten Sichten zu vereinen. Welche Teile verwendet werden, ergibt sich aus den Anforderungen und Kriterien, die in Abschnitt 2.2 und 3.5 aufgeführt werden. In den folgenden Abschnitten wird für die einzelnen Bereiche dargestellt, welche Aspekte Eingang in das Modell finden und welche nicht berücksichtigt werden.

5.1.1 Unternehmensmodellierung

Aus der Unternehmensmodellierung wird vor allem die nicht auf bestimmte Systeme fokussierte, unternehmensweite Sicht auf Geschäftsprozesse übernommen. Dies erfordern Kriterium 1 (Gleichartige Abbildung) und Kriterium 2 (Komplette Abbildung). Das heißt, das Modell der integrierten Architektur soll Elemente bereitstellen, die Geschäftsprozesse der Unternehmung darstellen. Dazu gehört auch die Bereitstellung von Elementen zur Abbildung der Organisationsstruktur,

die mit der Ausführung von Geschäftsprozessen verbunden ist.

Nicht berücksichtigt werden hingegen die Elemente der Unternehmensmodellierung, die Ziele oder Zweck von Unternehmungen, Prozessen oder Organisationen beschreiben, sowie Elemente, die strategische, planerische oder kontrollierende Aspekte modellieren. Dies würde der Anforderung widersprechen, daß die integrierte Architektur nur den Teil der Geschäftswelt modelliert, der durch einen Computer unterstützt werden soll. Dies schränkt auch die oben gemachte Aussage ein: Es werden nicht alle Geschäftsprozesse einer Unternehmung modelliert, sondern nur diejenigen, die auf eine Software abgebildet werden sollen.

5.1.2 Systementwicklungsprozeß

Systementwicklung steht hier für die aktuellen Softwareentwicklungsmethoden und deren Sichten auf ein Softwaresystem. Da diese Sichten je nach Paradigma unterschiedlich sind, beschränkt sich die Diskussion hier auf die objektorientierten Methoden, wie zum Beispiel der Unified Process [JBR98] mit UML [BRJ99] oder Catalysis [DW99].

Wie schon erwähnt, sind diese Methoden vorrangig dazu gedacht, einen konventionellen Softwareentwicklungsprozeß – unterteilt in Analyse, Design und Implementierung – zu unterstützen, der als Ergebnis ein Softwaresystem hat. Deshalb spielen hier viele Elemente eine Rolle, die für eine unternehmensweite, infrastrukturunabhängige Sichtweise nicht geeignet sind. Schon genannt wurden die Use Cases, andere Beispiele sind Komponenten (im Sinne von Teilen eines Softwaresystems) und Deployment Diagramme. Grundsätzlich werden solche Elemente nicht verwendet, die über eine logische Sicht auf ein fachliches Modell hinausgehen.

Die klassischen Konzepte und Elemente der Objektorientierung, die den derzeitigen Entwicklungsmethoden für Informationssysteme als Basis dienen, sind auch Basis der Modellelemente der integrierten Business Objects Architektur. Klassen und Relationen beschreiben die Business Objects und ihre Beziehungen untereinander. Diese Konzepte werden über die Meta Object Facility (MOF), einem Meta-Metamodell der OMG, in das in dieser Arbeit entwickelte Modell eingebracht. Die MOF wird im Detail in Abschnitt 5.3 beschrieben.

5.1.3 Infrastruktur

Die Sichten auf die Infrastruktur sind größtenteils produktabhängig. Auch neuere, als Basis für Softwaresysteme gedachte Standardarchitekturen wie die Object Management Architecture oder Enterprise JavaBeans (EJB) erfordern die Einführung von nicht-fachlichen Elementen. So könnte man durchaus die Entity Beans des EJB-Standards als Business Objects bezeichnen. Welche Business Objects (Entity, Process, Workflow oder Activity) jedoch wie auf ein Entity Bean abgebildet werden und wie diese dann voneinander abhängen und interagieren, wird nicht geklärt. So bleibt auch der EJB-Standard nur eine technische Architektur, wenn auch auf einer höheren Abstraktionsebene wie zum Beispiel eine universelle Programmiersprache.

Die Infrastruktur und damit die Softwarearchitektur wird im nachfolgend vorgestellten Modell nicht berücksichtigt. Das ergibt sich aus dem Kriterium 7 (Partitionierung) in dem gefordert wird, daß Business Objects so partitioniert werden, daß die eigentliche Geschäftslogik von der Anwendungslogik (und damit von der Infrastruktur) getrennt wird.

5.2 Grundlegende Designentscheidungen

Die in den vorangegangenen Abschnitten aufgezählten Gründe für oder wider die Aufnahme von Elementen in das Metamodell für eine integrierte Business Object Architektur ließen sich von den im Abschnitt 3.5 genannten Kriterien ableiten. Darüberhinaus wurden weitere Designentscheidungen für das Modell getroffen, die nachfolgend diskutiert werden.

5.2.1 Unternehmensweit

Die Forschungen zu den Business Objects wurden vorrangig durch zwei Probleme der Entwicklung von Softwaresystemen bestimmt:

5.2.1.1 Mehrfachverwendbar

Wie kann man es erreichen, daß fachliche Komponenten mehrfachverwendbar sind (siehe Definition 2.16 (Mehrfachverwendung))? Diese Fragestellung betont

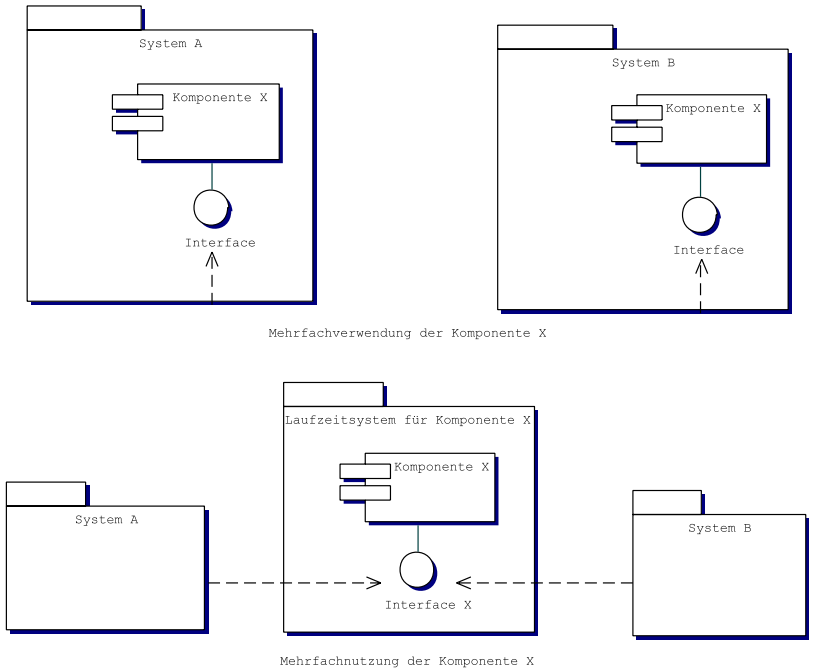

Abbildung 5.2: (Unternehmensweite) Mehrfachnutzung und (unternehmensübergreifende) Mehrfachverwendung

vor allem die Sicht der Softwareindustrie: Es sollen fachliche Komponenten, einmal entwickelt, an viele Firmen verkauft werden.

Die Mehrfachverwendung technischer Komponenten, wie GUI-Frameworks oder Netzwerkbibliotheken sind zumindest innerhalb einer Programmiersprache heute üblich. Bestes Beispiel sind die umfangreichen Frameworks, die als Standard für Java definiert wurden [GY96] und von kaum einem System, das in Java implementiert ist, nicht genutzt werden.

Wie schon in Kapitel 2 diskutiert, scheitert dieses Vorhaben zur Zeit noch an der Verfügbarkeit einheitlicher, normalisierbarer Verhaltensbeschreibungen. Deshalb fokussiert sich die Arbeit auf eine unternehmensweite Mehrfachnutzung von Business Objects. In Abbildung 5.2 werden die Unterschiede von Mehrfachnutzung und Mehrfachverwendung einer Komponente schematisch gegenübergestellt.

Die Object Management Architecture (OMA) erlaubt unabhängig von der Programmiersprache Dienste von verteilten Objekten mehrfachzuverwenden. Sie unterstützt dadurch eine unternehmensweite Nutzung von Diensten. Allerdings sind die Dienste, die in der OMA zur Zeit definiert sind, nur technisch, nicht fach-

lich. Unternehmensweit mehrfachnutzbare fachliche Komponenten bieten die in Abschnitt 3.3 beschriebenen Frameworks SAP BAPI und IBM San Francisco. Allerdings mit den genannten Einschränkungen, daß diese Frameworks nicht offen sind im Sinne der Definition 2.18 (offenes Informationssystem) sind.

5.2.1.2 Integration

Mit der Prozessorientierung, die spätestens seit dem Erscheinen des in Kapitel 2 vorgestellten Buches von Hammer und Champy und dem darin propagierten (Business-) Reengineering Paradigma [HC93], wurden sich viele Unternehmen bewußt, daß sich bei ihnen im Laufe der Zeit eine heterogene Softwarelandschaft entwickelt hat, die eine Integration von verschiedenen Systemen oder gar mit Systemen der Zulieferer bzw. Kunden unmöglich machte [HC93].

Verantwortlich dafür ist, daß die Systeme innerhalb Funktions- oder Organisationsstrukturen des Unternehmens in Auftrag gegeben wurden, sich dabei aber nicht nach den übergreifenden Prozessen gerichtet wurde. In einer heterogenen Softwarelandschaft werden diese Prozesse von verschiedenen Systemen auf unterschiedliche Art mehrfach implementiert [HC93]. Eine Mehrfachnutzung wurde dabei nicht angestrebt. Das heißt, fachliche Komponenten existieren in unterschiedlicher Granularität und Ausprägung mehrfach. Dies soll durch den Einsatz von Business Objects verhindert werden.

5.2.1.3 Unternehmensweit statt Unternehmensübergreifend

Die in Kapitel 3 und 4 vorgestellten Business Objects Modelle versuchen Mehrfachverwendung und Integration von fachlichen Komponenten zu ermöglichen. Es gibt zu den angesprochenen Problemen eine weitere Dimension: Soll mithilfe des Modells ein unternehmensweites Business Object Modell entwickelt werden, wie bei [Mar00, Tay94, ES98] oder ein unternehmensübergreifendes, wie bei [HS99, AF98] oder eines das beide Aspekte berücksichtigt, wie [OMG98a]?

Unternehmensübergreifend heißt, daß bei einem Business Object Modell die modellierten Business Objects, ihre Abhängigkeiten untereinander, sowie ihre Funktionalität für mehrere Unternehmen bzw. eine Domäne (wie zum Beispiel die Kontoverwaltung) standardisiert werden. Damit soll erreicht werden, daß sich ein Markt von wiederverwendbaren fachlichen Komponenten etabliert, in dem ein Komponenten-Typ (zum Beispiel das Business Object „Konto“) von verschiedenen Firmen hergestellt wird, wobei diese Komponenten untereinander austausch-

bar seien sollen.

Eine Austauschbarkeit sicherzustellen stellt jedoch erhebliche Anforderungen an die Mächtigkeit eines Modells. Denn es müssen nicht nur Struktur und Verbindungen eines Business Object Geflechts, sondern wie oben bereits erwähnt auch das genaue Verhalten der Geschäftslogik spezifiziert werden. Die Business Object Modelle, die zur Zeit einen unternehmensübergreifenden Aspekt berücksichtigen, haben keine umfassende Möglichkeit der Verhaltensspezifikation. Am weitestgehend ist hier die BOCA der OMG, die für jede Operation eines Business Objects Vor- und Nachbedingungen spezifizieren läßt. Eine Protokollspezifikation oder eine Spezifikation des Verhaltens von Prozessen ist nicht möglich.

Das in dieser Arbeit entwickelte Metamodell für eine integrierte Business Object Architektur ist Bestandteil eines methodischen Frameworks zur Entwicklung offener, unternehmensweiter Anwendungen. Es soll einem Unternehmen ermöglichen, *seine* Geschäftsprozesse auf eine IT-Infrastruktur abzubilden. Auf eine umfassende Verhaltensspezifikation, die eine unternehmensübergreifende Anwendung ermöglichen würde, wurde verzichtet.

5.2.2 Business Architektur

Die Software-Architektur eines Systems kann aus verschiedenen Sichtwinkeln, die eine oder mehrere Aspekte eines Systems darstellen, betrachtet werden. In [Kru95] werden fünf solcher Sichten vorgestellt:

- In der *Nutzungsfallsicht* (Use Case View) wird das Verhalten eines Systems dargestellt, so wie es sich einem externen Nutzer darstellen soll. Die Nutzungsfallsicht steht am Anfang, aus sollen laut [Kru95] sukzessive die anderen Sichten abgeleitet werden.
- Die *Logische Sicht* (Logical View, in [BRJ99] in Design View geändert) bezieht sich auf die funktionalen Anforderungen eines Systems. Es ist ein abstrahiertes Design Modell und identifiziert die wichtigsten Packages, Subsysteme und Klassen eines Systems. Sie definiert im Wesentlichen das Vokabular und die Funktionalität eines Systems.
- Die *Implementierungssicht* (Implementation View) beschreibt den Aufbau und den Zusammenhang von statischen Softwaremodulen wie Quelltext, Datendateien, Komponenten und ausführbare Dateien. Damit werden Systemaufbau und Konfigurationsmanagement unterstützt.

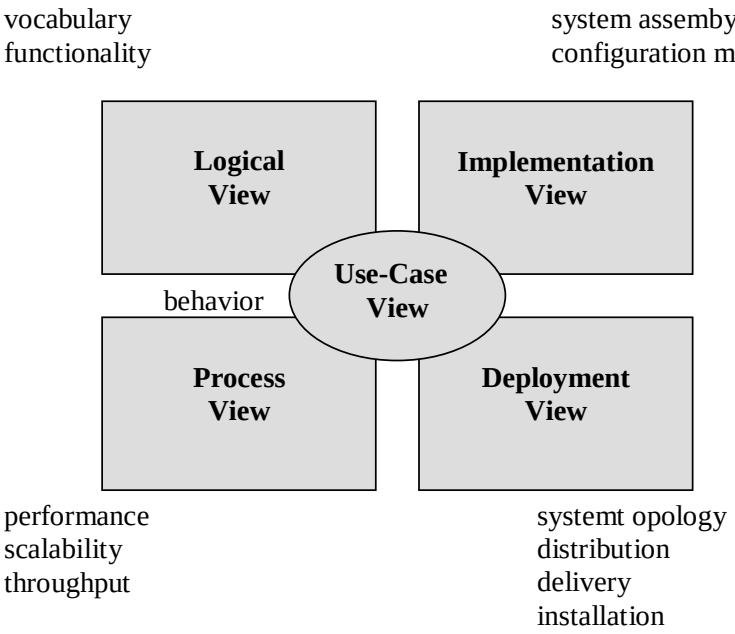

Abbildung 5.3: Das 4+1 Sichtenmodell der Softwarearchitektur (aus [Kru95])

- Aus der *Prozeßsicht* (Process View) werden Nebenläufigkeit, Parallelität, Systemstart und -ende sowie Verteilungsaspekte betrachtet. Aus dieser Sichtweise kann festgestellt werden, wie sich die Leistungsfähigkeit, Zuverlässigkeit und der Durchsatz eines Systems entwickeln wird.
- Die *Laufzeitsicht* (Deployment View)zeigt die verschiedenen ausführbaren Teile eines Systems und auf welchen Rechnerknoten sie ausgeführt werden. Mit dieser Sicht wird eine Systemtopologie dargestellt.

Dieses Sichtenmodell zeigt hier wieder die mehrfach erwähnte, Nutzungsfall- und System-zentrierte Vorgehensweise, die für die derzeitigen Softwaremethoden typisch sind. Für das im folgenden vorgestellte Metamodell einer integrierten Business Object Architektur entsprechen diese Sichten und die damit zusammenhängenden Konzepte nicht dem, was ausgehend von den Anforderungen und den Kriterien notwendig wäre.

Im Gegenteil, Implementierungssicht, Prozeßsicht und Laufzeitsicht sind genau die Sichten, auf die im Modell verzichtet werden soll. Aber auch Logische Sicht und Nutzungsfallsicht werden im Metamodell nicht im oben genannten Sinne integriert. Stattdessen beschränkt sich das Metamodell auf verschiedene Sichten auf eine Teilmenge der Business Architektur. Unter Business Architektur wer-

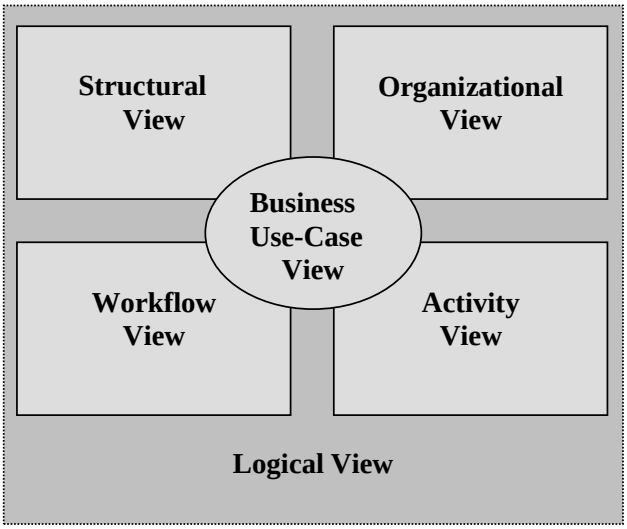

Abbildung 5.4: Das 4+1 Sichtenmodell des methodischen Frameworks

den dabei die Elemente der Geschäftswelt und deren Beziehungen untereinander verstanden. Teilmenge deshalb, weil nur diejenigen Aspekte der Geschäftswelt modelliert werden sollen, die auf einem Computer ausgeführt beziehungsweise verwaltet werden.

Die Sichten, die mit dem Metamodell ausdrückbar seinen sollen, sind:

- die *Struktursicht* (Structural View), in der die Business Object Ausführungsarchitektur beschrieben wird,
- die *Organisationssicht* (Organizational View), in der die Akteure eines Unternehmens und ihre Einordnung in die funktionalen wie organisatorischen Einheiten der Unternehmung, dazu die möglichen Rollen, die die Akteure einnehmen können, definiert wird,
- die *Vorgangssicht* (Workflow View), in der die dynamischen und hierarchischen Abhängigkeiten von arbeitsteiligen Vorgängen betrachtet wird,
- die *Aufgabensicht* (Activity View), in der die nicht arbeitsteiligen Aufgaben und die diese ausführenden Akteure beschreibt, sowie
- der *Geschäftsvorfallsicht* (Business Use Case View), die die Geschäftsvorfälle eines Geschäftsbereiches darstellt.

Die Struktur-, Organisations-, Vorgangs- und Aufgabensicht sind Teilsichten der oben dargestellten logischen Sicht des 4+1 Sichten-Modells aus [Kru95]. Sie alle

definieren das Vokabular und die Funktionalität eines Systems, nämlich der Unternehmung an sich. Weitere Sichten die Teilmengen einer logische Sicht sind, wären denkbar, so zum Beispiel einer Interaktions- oder Protokollsicht, die die dynamischen Abhängigkeiten oder die Protokolle, die zwischen Business Objects zulässig sind, darstellt. Wie schon im vorigen Abschnitt beschrieben, beschränkt sich das Metamodell bei der Beschreibung des Verhaltens auf die Modellierung der Abläufe der arbeitsteiligen Vorgänge. Die Operationen der Business Objects an sich werden nicht modelliert.

5.2.3 Standards

Die in dieser Arbeit entwickelte Methode ist gedacht für den Einsatz in mittleren bis großen Unternehmen und Organisationen und zwar von Seiten der Anwender, nicht der Entwickler. Das heißt eine Firma entscheidet sich, in Zukunft ihre Geschäftsprozesse mittels der Bestandteile des in dieser Arbeit vorgestellten Frameworks auf eine IT-Infrastruktur abzubilden. Die Methode ist weniger geeignet für Entwickler, das heißt zum Beispiel Softwarehäuser, die für Kunden Systeme entwickeln. Hier sind konventionelle Methoden besser geeignet, da es meistens darum geht, ein System in einem Projekt zu entwickeln.

Für die Akzeptanz auf Seiten der Anwender in den Unternehmen ist aber wichtig, daß mit der Methode nicht völlig neue, eigene Techniken zum Einsatz kommen in Bereichen, von denen nicht zwangsläufig die Verwirklichung der Grundkonzepte und -Gedanken der Methode abhängig sind. Zum Beispiel im Bereich Beschreibungstechniken: Hier hat sich in den letzten Jahren UML als Standardbeschreibungstechnik in den Unternehmen durchgesetzt. Mitarbeiter wurden geschult und teure Lizenzen für Modellierungswerkzeuge wie Rational Rose, Together oder Aonix Software Through Pictures gekauft. UML bietet eine große Menge unterschiedlicher Sichten auf und Diagrammarten für ein Modell. Dazu sind Möglichkeiten zur Erweiterung des UML Metamodells vorgesehen.

Gerade im Bereich der Workflow-Managementsysteme, der für diese Arbeit relevant ist, kommt jedes Produkt mit seiner eigenen Beschreibungstechnik auf den Markt, die keinem Standard entspricht. Nicht nur das dadurch schon von vornherein eine Integration mit anderen Werkzeugen erschwert wird, die Mitarbeiter müssen zusätzlich noch geschult werden.

Das in dieser Arbeit entwickelte Metamodell wird deshalb auf ein Profil der UML abgebildet. Die Beschreibung des Profils und die entsprechende Abbildung, werden in Kapitel 8 beschrieben. Diese Festlegung hat zunächst keine direkten Kon-

sequenzen für das Design des Metamodells. Sie beeinflußt aber die Art und Weise, wie das Metamodell spezifiziert wird.

Für die Spezifikation eines Metamodells gibt es mehrere Möglichkeiten:

- Mithilfe einer umgangssprachlichen Beschreibung. Hierbei ist die Gefahr jedoch groß, die einzelnen Elemente des Modells und ihre Abhängigkeiten sowie Constraints über dem Modell nicht eindeutig zu spezifizieren. Ein Beispiel hierfür sind die ersten Spezifikationen für CORBA [OMG92]. Sie waren nicht nur lückenhaft, sondern zum Teil nur umgangssprachlich formuliert, was dazu führte, daß die ersten Object Request Broker (ORB) Implementierungen nicht interoperabel waren, obwohl gerade das eines der wichtigsten Ziele der Object Management Architektur ist.

- Durch formale Methoden. Sie erlauben es, neben der Struktur – der Syntax – eines Modells auch das Verhalten – die Semantik – genau zu spezifizieren. So können nicht nur Aussagen über statische Eigenschaften eines Modells überprüft, sondern auch dynamische Eigenschaften verifiziert werden. Ein Beispiel für eine solche Methode ist FOCUS [BS01, RKB95] und darauf basierende Werkzeuge wie AutoFOCUS [HSSS96].

- Anwendung einer semiformalen Beschreibung. Semiformal heißt, das nur ein Teil des Modells formal spezifiziert wird. Dies ist in der Regel die statische Struktur – die Syntax – des Modells. Das Verhalten wird umgangssprachlich oder durch Referenzimplementierungen festgelegt. Beispiel ist die Spezifikation eines (Meta-) Modells auf Basis eines (Meta-) Metamodells.

Eine Spezifikation eines Modells durch eine formales Modell ist vom Standpunkt der Genauigkeit und Eindeutigkeit am sinnvollsten. Allerdings dies auch die aufwendigste Möglichkeit und bei umfangreicheren Modellen dann gerechtfertigt, wenn die Semantik eines Modells eindeutig definiert werden soll, um bestimmte Eigenschaften verifizieren zu können.

Für das Metamodell einer integrierten Business Object Architektur ist - wie im vorigen Abschnitt beschrieben - vor allem die Business Architektur ein zentraler Bestandteil. Es wird, bis auf die arbeitsteiligen Vorgänge, überwiegend die Struktur eines Business Object Modells spezifiziert. Zudem erfordert das in Kapitel 2 geforderte und in Kapitel 9 beschriebene Werkzeugkonzept, daß verschiedene Tools das Business Object Modell bearbeiten bzw. austauschen. In einem solchen Fall ist die Verwendung eines Meta-Metamodells zur Spezifikation geeignet.

Bei der Auswahl eines geeigneten Meta-Metamodell spielt nun die oben diskutierte Entscheidung für die UML als Beschreibungssprache eine Rolle: Diese wird mithilfe des Meta-Metamodells MOF, der Managed Object Facility der OMG spezifiziert [OMG97b]. Die MOF ist ein OMG-Standard und wurde zur Spezifikation von Metamodellen in verschiedenen Bereichen vor allem im Hinblick auf die Verwaltung von Modellen durch Tools und den Austausch von Modellen und deren Instanzen zwischen Tools definiert. Die MOF wird deshalb in dieser Arbeit als Meta-Metamodell zur Spezifikation des Business Object Metamodells verwendet. Im folgenden Abschnitt werden die Elemente und Eigenschaften der MOF beschrieben.

5.3 Managed Object Facility

Grundlegende Aufgabe eines Metamodells ist es, die Sprache – das heißt die Konzepte der zu modellieren Domäne und deren Abhängigkeiten untereinander – festzulegen, in der ein Modell formuliert werden kann. Diese Festlegungen werden durch die Sprache des Metamodells beschrieben. Diese Sprache kann wieder durch ein Metamodell (dem Meta-Metamodell) festgelegt werden.

5.3.1 Die MOF als Metamodell Framework

Im Prinzip läßt sich diese Vorgehensweise beliebig fortsetzen. Es reicht jedoch aus, bis zu einem Meta-Metamodell zu gehen [MN88], und weitere Meta-Ebenen dadurch überflüssig zu machen, daß die Sprachmittel des Meta-Metamodells mächtig genug sind, um sich selbst zu beschreiben [OMG97b].

Die MOF geht deshalb von einem Framework mit vier Ebenen zur Metamodellierung aus. In Abbildung 5.5 wird dieses Framework dargestellt. Die vier Ebenen haben folgende Aufgabe:

- Das in Ebene M3 befindliche MOF Modell definiert Modellierungskonstrukte, die zur Formulierung verschiedenster Metamodelle – und des MOF Modells selbst – verwendbar sind. Als Beispiel sind in der Abbildung die Elemente `MetaClass`, `MetaAssciation` und `MetaAttribute`. Die Hauptelemente des Modells werden im folgenden genauer vorgestellt.
- In der Ebene M2 befinden sich die Metamodelle. Sie stellen Ausdrucksmittel für die Formulierung von Modellen für unterschiedliche Anwendungs-

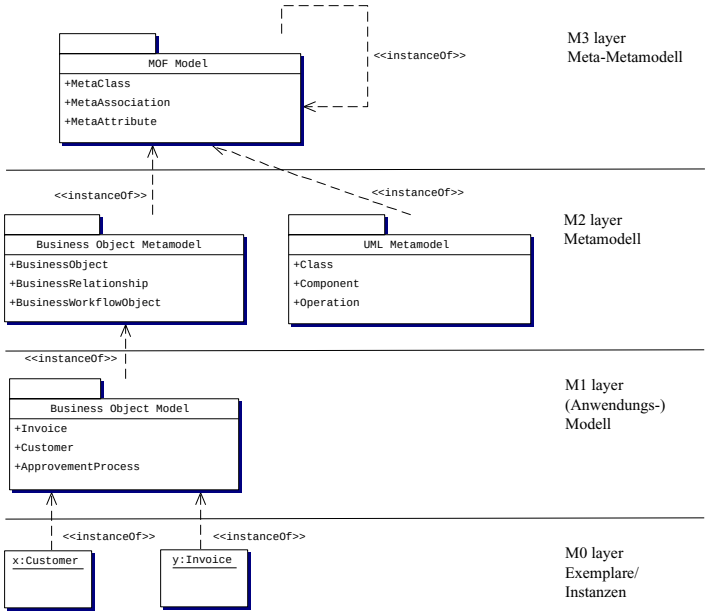

Abbildung 5.5: Beziehungen zwischen den vier MOF Modellebenen

modelle bereit. So zum Beispiel `BusinessObject` in einem Business Object Metamodell oder `Class` im UML Metamodell. Die Metamodelle in Ebene M2 sind Instanzen des MOF Modells der Ebene M3, daß heißt, sie wurden mit den Sprachmitteln des Meta-Metamodells definiert. So sind beispielsweise `BusinessObject` und `Class` Instanzen von des Meta-Metamodell-Konstruktes `MetaClass`.

- Die Modell-Ebene M1 setzt sich aus den jeweiligen Anwendungsmodellen zusammen. Diese sind Instanzen der jeweiligen Metamodelle der Ebene M2. Die Anwendungsmodelle klassifizieren einen bestimmten Anwendungsbereich und beschreiben einen Ausschnitt der gemeinsamen Eigenschaften und Abhängigkeiten der Konzepte des durch sie modellierten Systems. Zum Beispiel `Invoice` und `Customer` als Instanzen der Metaklasse `BusinessObject`, sowie `ApprovementProcess` als Instanz der Metaklasse `BusinessWorkflow` des Business Object Metamodells.

- In der Ebene M0 schließlich findet man die konkreten Daten eines durch ein Modell definiertes System. Diese Daten sind Exemplare der Modellelemente des Anwendungsmodells. Zum Beispiel Kunde „x“ und Rechnung „y“ als Exemplare der Modellelemente `Customer` bzw. `Invoice`.

Neben der besseren Vergleichbarkeit und des leichteren Verständnis von Metamodellen für die Modellierer, liegen die größten Vorteile einer solchen Architektur in der Wiederverwendung von Werkzeugen, die in irgendeiner Form (Meta-) Modelle verwalten oder transformieren, sowie in der Interoperabilität und der gegenseitigen Abbildbarkeit von Metamodellen, die über ein gemeinsames Meta-Metamodell definiert wurden. In dieser Arbeit wird dieser Vorteil dadurch genutzt, daß das Business Object Metamodell, eine Instanz des MOF Modells, auf ein Profil des UML-Metamodells, ebenfalls eine Instanz des MOF Modells, abgebildet wird.

5.3.2 Elemente des MOF Modells

In diesem Abschnitt werden die wichtigsten Modellierungskonstrukte vorgestellt, das heißt, die Sprache, mit deren Hilfe Metamodelle definiert werden sollen.

Die MOF nutzt dazu ein objektorientiertes Modellierungsframework, das im Grunde eine Untermenge des UML Frameworks ist. Ein Vorteil dieser Tatsache ist, daß Metamodelle – definiert mithilfe des MOF Modells – damit durch eine vereinfachte Variante des Klassendiagramms der UML dargestellt werden können, wie es auch in dieser Arbeit geschieht.

In Abbildung 5.6 sind die meisten Elemente des MOF Modells dargestellt. Es gibt vier Kernelemente:

- Klassen (`Class`), die zur Modellierung der Metaobjekte eines Metamodells dienen,
- Relationen (`Association`) die binäre Verbindungen zwischen den Metaobjekten modellieren,
- Typen (`DataType`) die strukturierte Daten, die weder Klassen noch Relationen sind, modellieren, sowie
- Module (`Package`), die eine Aufteilung eines Modells ermöglichen.

Diese vier Elemente sind Generalisierbar (`GeneralizableElement`), das heißt, zwei durch eine Generalisierungsrelation (`Generalizes`) miteinander in Beziehung gesetzte Elemente gehen eine Supertyp-Subtyp-Verbindung ein. Der Inhalt (`containedElements`) des generalisierten Elements (Supertyp) wird

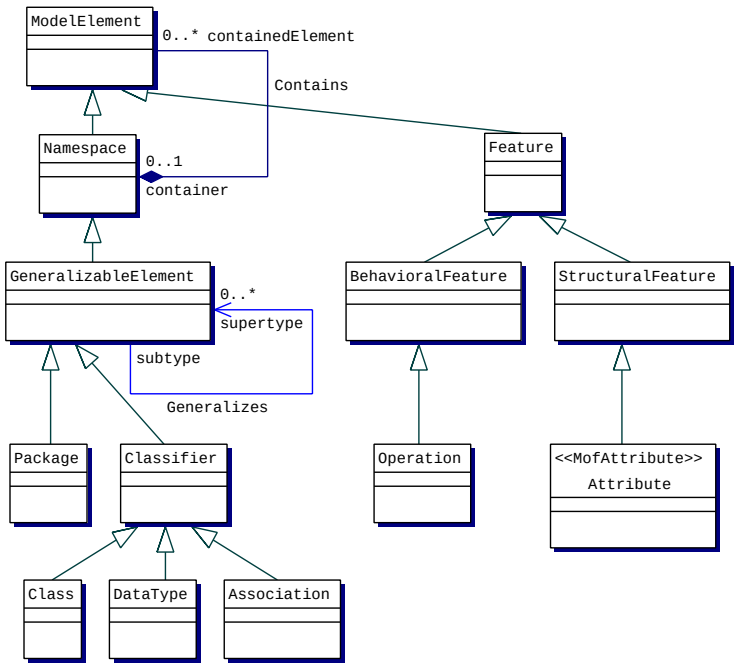

Abbildung 5.6: Elemente des MOF Meta-Metamodells (Ausschnitt)

damit dem Inhalt des Subtypen-Elements hinzugefügt. Inhalt heißt, das die generalisierbaren Elemente (Class, Association, Package, DataType) einen Namensraum (`Namespace`) darstellen, der andere Modellelemente (`ModelElement`) über eine Ist-Enthalten-in Beziehung (`Contains`) enthalten kann.

Neben diesen Elementen gibt es weitere Modellelemente, die Eigenschaftselemente (`Feature`). Es wird im MOF Modell zwischen Elementen unterschieden, die Verhalten modellieren sollen (`BehavioralFeature`) und Elemente die strukturelle Eigenschaften beschreiben (`StructuralFeature`). Nur die Elemente, die Verhalten modellieren bilden einen Namensraum, beide sind nicht generalisierbar. Die Verbindung mit den vier oben genannten Elementen geschieht über die Inhaltsbeziehung in der Form, daß die Eigenschaftselemente im Namensraum eines anderen Elementes enthalten sein können.

5.3.3 Object Constraint Language

Die im vorigen Abschnitt vorgestellten Modellelemente des MOF-Modells geben ein Grundgerüst für die Definition von Metamodellen vor. In der dargestellten Form ist es aber nicht verwendbar, denn es läßt zuviele nicht erwünschte Eigen-

schaften zu. So könnte zum Beispiel ein Modellelement sich selbst enthalten oder ein Subtyp in einer Generalisierungsbeziehung gleichzeitig sein eigener Supertyp sein.

Um die möglichen M2-Modelle, die sich mit den Elementen des MOF-Modells definieren lassen, auf diejenigen einzuschränken, die sinnvoll sind, werden zusätzlich zu den oben genannten Elementen Regeln definiert. In der Version 1.3 [OMG99b] der MOF Spezifikation sind 70 solcher Regeln festgelegt.

Die Definition der einschränkenden Regeln erfolgt mithilfe der Object Constraint Language (OCL), die Teil der UML Spezifikation [OMG99a] der OMG ist. Die OCL ist eine formale Sprache zur Formulierung von Regeln über einem Modell. Sie hat ihren Ursprung in der Syntropy Methode [CD94] und wurde von IBM als Sprache zur Definition von Business Regeln entwickelt [WK99]. Eine genaue Beschreibung des Sprachumfangs und Ausdrucksmöglichkeiten findet sich in [OMG99a]. Mit OCL können nur Ausdrücke beschrieben werden, die keine Seiteneffekte auf das Modell haben. Es ist nicht vorgesehen, das Verhalten von Modellelementen, speziell von Operationen, zu implementieren.

OCL wird in der UML Spezifikation verwendet, um folgende Ausdrücke zu spezifizieren:

- Invarianten für Klassen und Typen eines Modells,
- Typengebundene Invarianten für Stereotypen,
- Vor- und Nachbedingungen für Operationen und Methoden,
- Wächteranweisungen,
- Navigationsanweisungen und
- Einschränkungen auf Operationen.

OCL ist weder auf die UML, noch auf die MOF beschränkt, sondern kann durch Anwendung als Ausdrucksmittel für Syntaxregeln für jedes MOF-basierte Metamodell verwendet werden. Das Business Object Metamodell, das im folgendem Abschnitt schrittweise erstellt wird, wird OCL als Sprache zur Formulierung von Einschränkungen über dem Modell verwendet.

5.4 Zusammenfassung

In diesem Kapitel wurden die Entwurfsgrundsätze für das Metamodell des methodischen Frameworks festgelegt. Neben funktionalen Eigenschaften wie der Auswahl der zu modellierenden Elemente der Geschäftswelt wurden auch nichtfunktionale Eigenschaften wie die Verwendung des MOF-Modells als Meta-Metamodells bestimmt. Diese Festlegungen bestimmt den Aufbau und die Struktur des nachfolgend vorgestellten Metamodells maßgeblich.

Im nächste Teil der Arbeit werden nun die Bestandteile des methodischen Frameworks kapitelweise vorgestellt. Begonnen wird dabei mit der grundlegenden Basis des Frameworks, dem Business Object Metamodell.

Teil III

Methodisches Framework

Kapitel 6

Business Object Metamodell

In diesem Kapitel wird schrittweise ein Business Object Metamodell entwickelt, das einen integrierten Entwurf und eine unternehmensweite Realisierung von Informationssystemen ermöglicht. Das Metamodell ist dabei der zentrale Bestandteil des in dieser Arbeit entwickelten Frameworks. Das vorgestellte Metamodell erfüllt die im vorigen Teil der Arbeit diskutierten Kriterien, insbesondere die Integration von arbeitsteiligen Vorgängen und der Unternehmensorganisation und richtet sich nach den dort aufgestellten Entwurfsgrundsätzen.

Die wesentlichen, neuen Eigenschaften, die dieses Metamodell von bisherigen Ansätzen unterscheiden sind:

- Vollständige Integration von arbeitsteiligen Vorgängen in das Modell.
- Konzentration und Berücksichtigung der unternehmensweiten Mehrfachnutzung von Business Objects.
- Durchgängige und einheitliche Modellierung aller Elemente der Geschäftswelt ohne Brüche und ohne Abhängigkeit zur Infrastruktur.
- Verwendung des standardisierten MOF Metamodells als Spezifikationssprache für das Metamodell.
- Aufteilung des gesamten Modells in Packages mit der Möglichkeit, Erweiterungen einfach einzubinden.

6.1 Vorgehen

Im den folgenden Abschnitten wird das Business Object Metamodell vorgestellt, das Grundlage für das Framework zur Entwicklung unternehmensweiter Anwendungen ist. Die Anforderungen und Kriterien, die diesem Metamodell zu Grunde liegen wurden im Kapitel 2 und 3 besprochen. Hauptsächlich sind dies:

- Das Metamodell soll die Abbildung von Teilen einer Geschäftswelt auf Modellelemente erlauben und zwar in der Form, das die Modellelemente in sich geschlossene Konzepte der Geschäftswelt definieren.
- Die Abhängigkeiten die zwischen den Konzepten der Geschäftswelt sollen mit Elementen des Metamodells ausdrückbar sein. Abhängigkeiten zu Elementen außerhalb der Geschäftswelt sind zu vermeiden.
- Die Modelle, die auf Basis des Metamodells definiert wurden, sollen (durch eine automatisierte Abbildungsfunktion) auf einer IT-Infrastruktur ausführbar sein. Dabei soll das Modell unabhängig bleiben von den Anforderungen, die die IT-Infrastruktur stellt.

Das Business Object Metamodell soll nicht dazu dienen, wie schon in Abbildung 2.4 gezeigt, wie in traditionellen Methoden ein Modell eines Informationssystems zu entwickeln, dessen modellierter Ausschnitt der Geschäftswelt bezogen ist nur auf die Anforderungen (Use Cases) des Systems und zudem noch die Abbildung auf die Infrastruktur mitmodelliert.

Bei Entwicklung des Modells wird schrittweise vorgegangen: Zunächst werden Basiselemente des Metamodells eingeführt (Package „Base“). Diese entsprechen im wesentlichen den Kernkonzepten der MOF, allerdings auf Ebene des Metamodells (Metaebene M2) und des UML Metamodells. Die Beschreibungen und Regeln für diese Elemente wurde, falls möglich, in dieses Metamodell aus [OMG99b] sinngemäß übernommen. Aufbauend auf diese Elemente wird das Metamodell um diejenigen Elemente erweitert, die die grundsätzlichen strukturellen und dynamischen Konzepte der Geschäftswelt darstellen (getrennt in die Packages „Structure“ und „Dynamics“). Und schließlich werden Elemente zur Beschreibung arbeitsteiliger Vorgänge der Geschäftswelt in das Modell integriert.

6.2 Allgemeine Modellelemente

Die im folgenden vorgestellten allgemeinen Modellelemente bilden die Basis des Business Objects Metamodells. Die meisten dieser Modellelemente sind abstrakt, das heißt, sie bestimmen gemeinsame Eigenschaften und Konzepte von konkreten Modellelementen, werden während der Modellierung aber nicht explizit sichtbar.

Die allgemeinen Modellelemente sind vor allem notwendig, um einen einheitlichen, strukturierten Zugriff auf Eigenschaften von Modellelementen durch Modellierungswerkzeuge zu gewährleisten. Das diese Modellelemente im wesentlichen den MOF-Basiselementen entsprechen hat den Vorteil, daß alle Werkzeuge, die MOF-konforme Metamodelle verwalten (z.B. alle UML-Werkzeuge) genutzt werden können, um auch Modelle auf Basis des Business Object Metamodells bearbeiten zu können. In Kapitel 9 wird gezeigt, wie das Werkzeug „Together“ angepaßt wird, um auf Basis des Business Object Metamodells modellieren zu können.

6.2.1 Basis

Die Basiselemente des Metamodells dienen der Einführung folgender Konzepte für das Business Object Modell:

- Klassifizierung
- Generalisierung
- Namensräume
- Typen
- Eigenschaften

In Abbildung 6.1 ist das Basismodell als UML-Diagramm dargestellt. Wie schon erwähnt, ist dies auch gleichzeitig (bei Beschränkung der Ausdrucksmittel) eine MOF-Definition des Metamodells. So ist das Klassensymbol für `ModelElement` eine graphische Darstellung für `MOF::Class ModelElement`.

Abweichend vom UML-Metamodell oder MOF-Modell, die dieselben Modellelemente in ihren Basismodellen haben, befinden sich im Business Object Metamodell keine konkreten Modellklassen im Package „Base“. Konkrete Modellklassen

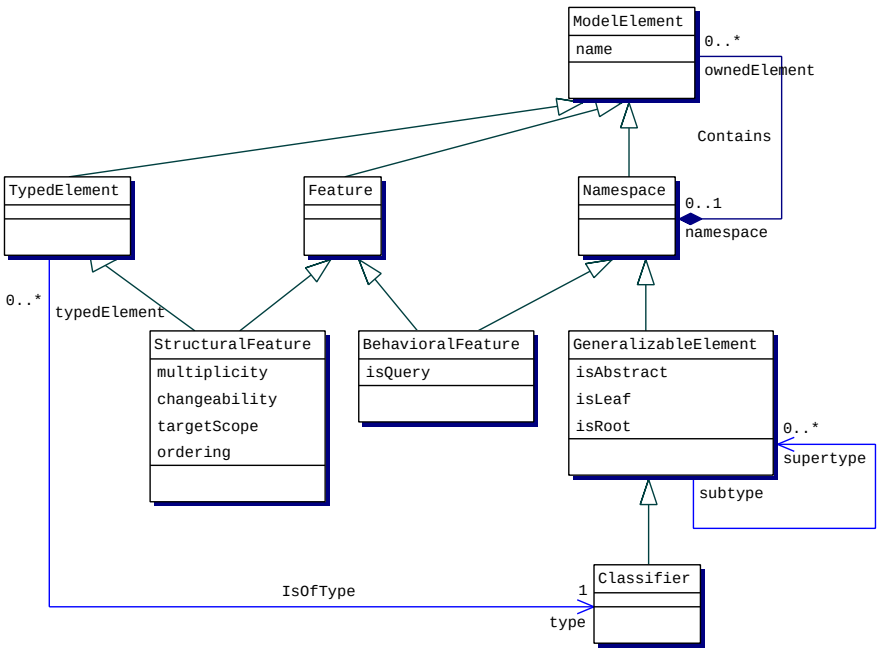

Abbildung 6.1: Basiselemente des Business Object Metamodells (Package `Base`)

sind diejenigen Klassen, mit denen die abstrakte Syntax zum Ausruck des Modells bilden. Beim Business Object Metamodell werden konkrete Klassen nur in den beiden Packages „Structure“ und „Dynamics“ definiert.

Im Folgenden werden die Modellelemente des Basismodells in folgender Form dargestellt:

- Name des Modellelements
- Beschreibung
- Attribute des Modellelements (falls vorhanden)
- Referenzen zu anderen Modellelementen (falls vorhanden)
- Regeln für dieses Modellelement (falls vorhanden)

Da das Metamodell inkrementell eingeführt wird, können sich Regeln in den späteren Packages wieder auf ein Basiselement beziehen. Hilfsregel befinden sich im Anhang.

6.2.1.1 ModelElement

Die abstrakte Klasse `ModelElement` ist die oberste Klasse im Metamodell, sie ist eine Generalisierung jeder anderen Klasse. Sie klassifiziert alle elementaren Konstrukte des Modells.

Attribute

name Jedem Modelelement des Modells wird ein Name zugewiesen. Hierbei handelt es sich um einen Ebene M1 Namen, das heißt, er ist kein Name eines Ebene M2 Elementes. Stattdessen wird dieses Attribut verwendet um den Bezeichner der M1 Instanzen des jeweiligen Modellelementes festzulegen.

Referenzen

namespace Bezeichnet den Namensraum, in dem sich das Element befindet. Bei `Contains` handelt es sich um eine Kompositions-Beziehung, das heißt, jedes Modellelement kann sich nur in einem Namensraum befinden. Sie wird durch `Namespace::namespace` definiert.

Regel 1 (Module als Wurzel) *Alle Modellelemente müssen einem Namensraum zugeordnet sein. Ausnahme sind* `Domains`*. Sie werdem im Package „Structure“ eingeführt. Sie sind also die Modellelemente, die einen oder mehrere Wurzelknoten des Modell-Namensraumes bilden.*

```
context ModelElement
inv:
  not self.oclIsTypeOf(Domain) implies
    self.namespace -> size = 1
```

6.2.1.2 Namespace

`Namespace` ist eine abstrakte Basisklasse für alle Modellelemente, die andere Modellelemente enthalten können. Zugleich wird ein Namensraum definiert, mit dem zugleich die Menge der möglichen Namen für die enthaltenen Modellelemente eingeschränkt werden können. Die Klasse erfüllt damit zwei Aufgaben: Die dient dazu, im Modell Namensräume und Ist-Enthalten-in Beziehungen zwischen Modellelementen bereitzustellen.

	Domain	BEO	BSO	BWO	BTO	BAO	DataType	Association	Attribute	Reference	Operation	Event	Parameter	AssociationEnd	Proxy
Domain	•	•	•	•	•	•	•	•							•
BEO							•		•	•	•	•			
BSO							•		•	•	•	•			
BWO		•		•	•	•	•		•	•	•	•			
BTO							•		•	•	•	•			
BAO							•		•	•	•	•			
DataType													•		
Association														•	
Operation							•						•		
Event							•		•				•		
Proxy															

Tabelle 6.1: Übersicht über die `Contains`-Beziehung des Metamodells

In Tabelle 6.1 wird die wichtige `Contains` Beziehung zwischen den Elementen des Metamodells in einer Übersicht dargestellt. In der Vertikalen sind alle konkreten Modellelemente des Business Objects Metamodells aufgeführt, die überhaupt andere Elemente in ihrem Namensraum beinhalten können. In der Horizontalen sind alle Modellelemente aufgeführt, die in diesen Namensräumen enthalten sein können. Eine ausführliche Beschreibung der einzelnen Modellelemente erfolgt in den nächsten Abschnitten.

Referenzen

contents Die Menge der direkt im Namensraum befindlichen Modellelemente. Sie wird durch `ModelElement::ownedElement` definiert. Diese Menge enthält auch andere Namensräume, nicht jedoch die Modellelemente, die sich in den in diesen Namensräumen befinden.

Regel 2 (Eindeutiger Name) *Der Name eines Modellelementes muß in einem Namensraum eindeutig sein.*

```
context Namespace
inv:
```

```
self.contents.forAll(e1, e2 |
  e1.name = e2.name implies e1 = e2)
```

6.2.1.3 GeneralizableElement

Die abstrakte Klasse `GeneralizableElement` ist die Basis für alle Modellelemente, die durch eine Generalisierungsrelation miteinander verknüpft werden können. Diese Relation weist einem der beteiligten Modellelement die Rolle `subtype`, den anderen Modellelement die Rolle `supertype` zu. Das Element mit der Rolle `subtype` übernimmt dabei alle Eigenschaften `Features` der anderen Elemente und zusätzlich alle Eigenschaften, die diese Elemente selbst in einer Generalisierungsrelation übernehmen.

Die übernommenen Eigenschaften werden in den Namensraum des Elementes mit der Rolle `subtype` eingefügt. Dies bedeutet aber nach Regel 2, daß eine Generalisierungsrelation zwischen Elementen, die gleiche Namen für Elemente ihrem Namensraum haben, nicht möglich ist.

Attribute

isAbstract Dieses Attribut legt fest, ob ein `GeneralizableElement` Instanzen in Ebene M1 haben darf, oder nicht. Wenn nicht, dann wird vorausgesetzt, daß es keine Instanzen dieses Elements in Ebene M1 geben kann, außer es existiert in der Generalisierungshierarchie eine Spezialisierung dieses `GeneralizableElement`, dessen Attribut `isAbstract` nicht wahr ist.

isRoot Bestimmt, ob das Element eine Generalisierungsrelation in der Rolle `subtype` eingehen darf.

isLeaf Bestimmt, ob das Element eine Generalisierungsrelation in der Rolle `supertype` eingehen darf.

Referenzen

supertypes Enthält die Menge aller Elemente, die die Rolle `supertype` in einer Generalisierungsrelation mit dem Element eingehen. Das heißt aber auch, die Elemente die wiederum eine Generalisierungsrelation mit den Elemente dieser Menge haben, sind nicht in `supertypes` enthalten.

Regel 3 (Keine Zyklen) *Ein* `GenerizableElement` *kann weder direkt noch indirekt sein eigener* `supertype` *sein.*

```
context GeneralizableElement
inv:
  self.allSupertypes() -> forAll(s | s <> self)
```

Regel 4 (Gleicher Typ) *Ein* `supertype` *eines* `GeneralizableElements` *muß vom selben Typ (Typ auf Ebene M2) sein.*

```
context GeneralizableElement
inv:
  self.supertypes ->
       forAll(s | s.oclType() = self.oclType())
```

Regel 5 (Supertyp Namen) *Die Namen des Namensraumes des* `GeneralizableElements` *dürfen nicht gleich den Namen aus den Namensräumen der direktem oder indirekten* `supertype` *Elementen sein.*

```
context GeneralizableElement
inv:
  let superContents = self.allSupertypes() ->
    collect(s | s.contents) in
  self.contents ->
    forAll(n1 | superContents ->
      forAll(n2 | n1.name = n2.name implies m1 = m2))
```

6.2.1.4 TypedElement

Die abstrakte Klasse `TypedElement` ist ein Element zur Abstrahierung von Modellelementen, die einen Typ als Teil ihrer Definition benötigen. Dabei definiert das Element den Typ nicht selbst, sondern ist dazu verbunden mit einen `Classifier`.

Referenzen

type Ist ein Verweis auf den Typen, für das durch `TypedElement` dargestellte Element.

6.2.1.5 Classifier

Die abstrakte Klasse `Classifier` dient der Klassifizierung von Instanzen (auf Ebene M1). Die Klassifizierung basiert auf den Eigenschaften, die mit der Klasse `Features` definiert sind. Ein `Classifier` enthält (durch die `Contains` Beziehung eine Menge von `Features`. Diese Eigenschaften gelten damit für alle Ebene M1 Instanzen des Elements `Classifier`.

`Classifier` ist die Oberklasse für die in den späteren Abschnitten eingeführten Kernelemente eines Business Object Modells. Im UML-Metamodell ist `Classifier` die Generalisierung für die Kernkonzepte objektorientierter Modelle: Klassen, Assoziationen, Interfaces, Datentypen und Komponenten.

6.2.1.6 Feature

Ein Modellelement der abstrakten Klasse `Feature` dient dazu, bestimmte Eigenschaften anderer Modellelemente zu bestimmten. Ein Modellelement wird durch Eigenschaften beschrieben, wenn es `Feature` Elemente in seinem Namenraum enthält. Dies gilt insbesondere für `Classifier`, die durch die in ihrem Namensraum enthaltenen `Feature` Elemente die Eigenschaften einer ganzen Klasse von Objekten ihrer Ebene M1 Instanzen festlegen.

Attribute

scope Definiert, ob es eine Instanz der Eigenschaft eines `Classifiers` für alle Instanzen dieses `Classifiers` gibt (`scope=classifier`), oder ob es für jede Instanz des `Classifiers` jeweils eine Instanz des `Features` gibt (`scope=instance`). In objektorientierten Modellen der UML zum Beispiel wird über dieses Attribut festgelegt, ob ein Attribut einer Klasse einen Wert für alle Objekte der Klasse, oder jedes Objekt seinen eigenen Wert für das Attribut hat.

6.2.1.7 StructuralFeature

Ein `StructuralFeature` stellt eine statischen bzw. strukturelle Eigenschaft eines Modellelementes dar. `StructualFeature` ist eine abstrakte Klasse. Von ihr wird die konkrete Modellklasse `Attribute` abgeleitet, die im Package „Structure“ eingeführt wird. Es ist eine Subklasse der Klasse

`TypedElement`. Das heißt, jedes `StructuralFeature` hat als Typ, der durch einen `Classifier` definiert wird.

Attribute

multiplicity Das Attribut `multiplicity` definiert Regeln für die Menge der Werte, die ein `StructuralFeature` enthalten kann. Zum einen die Kardinalität, daß heißt die untere und obere Schranke für die Anzahl der Werte. Andererseits wird festgelegt, ob die Menge der Werte geordnet ist. Darüberhinaus wird definiert, ob die jeweiligen Elemente der Wertemenge eindeutig sein müssen.

isChangeable Durch `isChangeable` wird festgelegt, ob die Werte, die ein `StructuralFeature` enthalten kann geändert werden können.

6.2.1.8 BehavioralFeature

Durch ein `BehavioralFeature` wird eine dynamische Eigenschaft eines Modellelementes, das dieses enthält, festgelegt. Ein `BehavioralFeature` ist auch gleichzeitig ein Namensraum und kann deshalb selbst durch Eigenschaften beschrieben werden. Konkrete Modellklassen, die von dieser abstrakten Klasse abgeleitet sind, werden im Package „Dynamics“ beschrieben.

6.3 Business Objects Modellelemente

Nachdem im letztem Abschnitt allgemeine Modellelemente eingeführt wurden, werden diese im nächsten Abschnitt um die – überwiegend konkreten – Modellelemente zur Modellierung von Business Object Systemen ergänzt. Aus Gründen der Übersichtlichkeit werden die Modellelemente zur Modellierung von arbeitsteiligen Vorgängen und Organisationsstrukturen erst im Abschnitt 6.4 hinzugefügt.

6.3.1 Struktur

In diesem Abschnitt werden die Modellelemente eingeführt, die zur Klassifizierung der strukturellen Elemente der Geschäftswelt dienen und damit hauptsäch-

lich das jeweilige Business Object Modell darstellen. Dies sind vorrangig Klassifizierungen für Phänomene der Geschäftswelt und deren Beziehungen untereinander.

In Abschnitt 3.2 wurden bereits verschiedene Klassifizierungen diskutiert. Das Business Object Metamodell sieht folgende Modellelemente zur Klassifizierung der fachlichen Konzepte der Geschäftswelt vor:

- Domains
- Business Entity Objects
- Business Service Objects
- Business Activity Objects (Abschnitt 6.4)
- Business Workflow Objects (Abschnitt 6.4)
- Business Subjects (Abschnitt 6.4)

Neben diesen Elementen werden Relationen zur Klassizierung verschiedener Beziehungen zwischen den Konzepten definiert.

Außer den fachlichen Konzepten werden noch Modellelemente für „neutrale“, also nicht fachlich, aber auch nicht technisch gebundene Konzepte, wie Datentypen, Attribute und Proxies eingeführt. Alle Modellelemente zur Modellierung von strukturellen Elementen der Geschäftswelt werden im Packages „Structure“ zusammengefaßt. In Abbildung 6.2 sind bereits die Modellelemente zur Modellierung von arbeitsteiligen Vorgängen dargestellt. Klassen aus anderen Packages („Base“ oder „Dynamics“) sind grau dargestellt. In den folgenden Abschnitten werden die Modellelemente diskutiert.

6.3.1.1 Domain

Ein unternehmensweites Modell besteht in der Regel aus vielen unterschiedlichen Business Objects. Sie gehören – das fordert das Business Object Paradigma – in eine einheitliche, unternehmensweite Ebene. Damit ein solches Modell nicht unübersichtlich wird, muß es entsprechend partitioniert werden. In dieser Arbeit werden `Domains` als Hilfsmittel hierfür eingeführt.

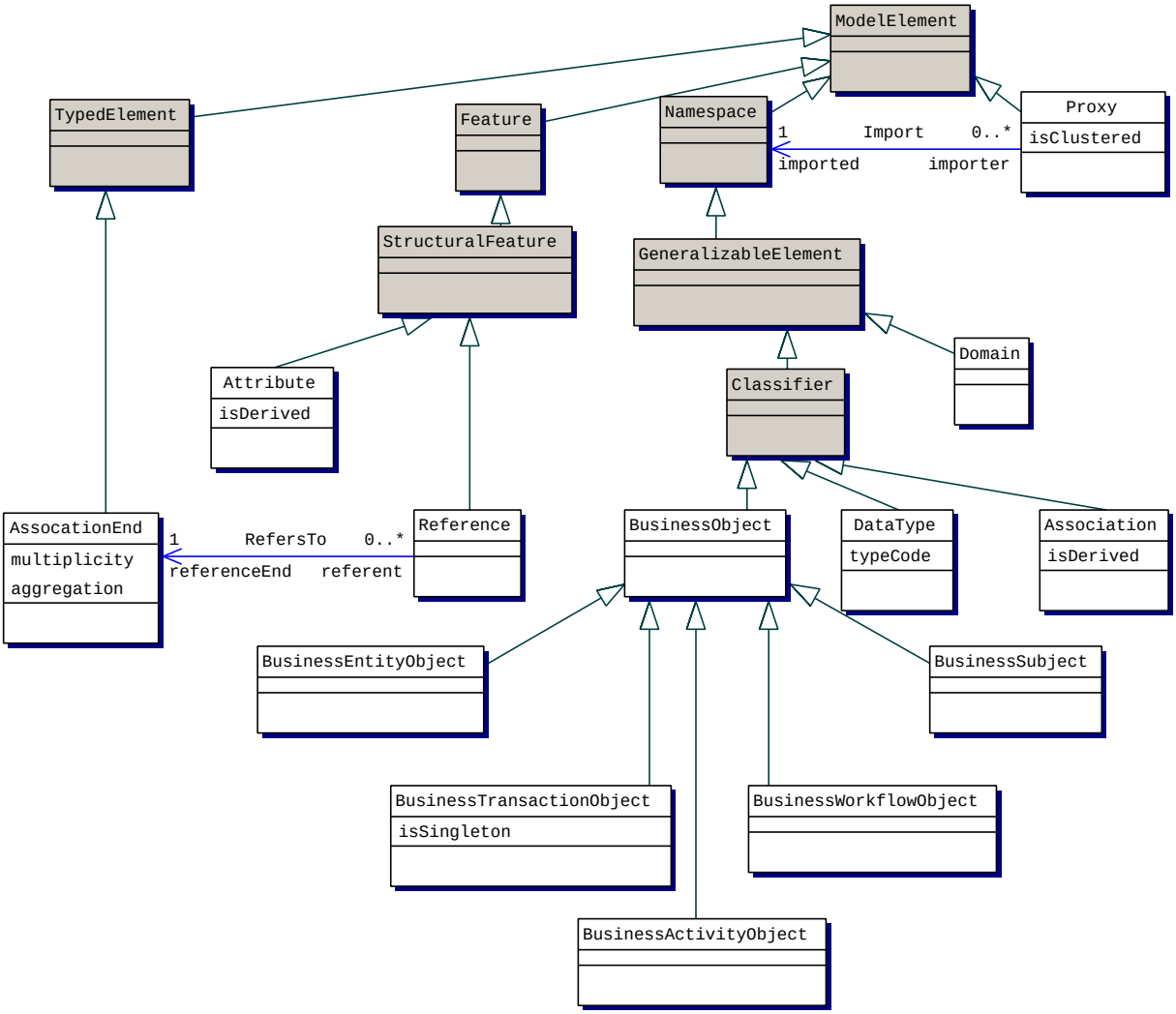

Abbildung 6.2: Strukturelemente Metamodells (Package `Structure`)

Eine `Domain` ist ein Geschäftsfeld der Unternehmung oder die Unternehmung selbst. Die Geschäftsfelder können wiederum in verschiedene Felder aufgeteilt werden, das heißt, `Domains` können andere `Domains` enthalten.

Im Vergleich mit dem Packagekonzept in UML gibt es zwei wesentlichen Unterschiede: Zum Einen gibt es keine Vererbungsbeziehungen zwischen Domains (Regel 7), zum Anderen ist die Trennung der `Domains` untereinander nicht strikt, denn es ist unrealistisch das alle Business Objects genau einem Geschäftsfeld zuzuordnen sind. Im Gegenteil, im Business Process Reengineering wird ja gerade versucht, die vorher fragmentierten Prozesse zu unternehmensweiten Prozessen durchgängig miteinander zu verknüpfen. Im Business Object Metamodell wird die strikte Trennung durch die `Proxy` Elemente aufgehoben, die im nächsten Abschnitt beschrieben werden.

Regel 6 (Modellelemente einer Domain) *In einer `Domain` können folgende Modellelemente enthalten sein: `Domain`, `BusinessEntityObject`, `BusinessServiceObject`, `BusinessWorkflowObject`, `BusinessSubject`, `DataType`, `Association`, `Event` und `Proxy`.*

```
context Domain
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(Domain) or
    e.oclIsKindOf(BusinessEntityObject) or
    e.oclIsKindOf(BusinessServiceObject) or
    e.oclIsKindOf(BusinessWorkflowObject) or
    e.oclIsKindOf(BusinessSubject) or
    e.oclIsKindOf(DataType) or
    e.oclIsKindOf(Association) or
    e.oclIsKindOf(Event) or
    e.oclIsKindOf(Proxy) )
```

Regel 7 (Keine Generalisierungsrelation für Domains) *Eine Generalisierungsrelation zwischen zwei `Domains` würde bedeuten, daß auf Ebene M1 die Instanz der abgeleiteten `Domain` eine Kopie der in der generalisierten `Domain` enthaltenen Modellelemente enthalten würde. Eine Kopie macht aber bei einem Modellelement, das ein und dasselbe Phänomen der Geschäftswelt modelliert, keinen Sinn. Deshalb ist keine Generalisierungsrelation zwischen `Domains` erlaubt.*

```
context Domain
```

```
inv:
  self.supertypes -> isEmpty and
  self.isRoot and
  self.isLeaf and
  not self.isAbstract
```

6.3.1.2 Proxy

Ein `Proxy` Modellelement blendet ein Modellelement aus einer anderen `Domain` in die eigene `Domain` ein. Das eingeblendete Modellelement stellt jedoch keine Kopie dar, das heißt die Ebene M1 Instanzen des Modellelementes und seiner `Proxies` sind identisch. Das heißt: Modellelemente sind unter verschiedenen Namen wie „de.firma.production.Part“ und „de.firma.sourcing.Part“ im Modell vorhanden, sind aber aus logischer Sicht *ein* Modellelement. Hier unterschiedet sich das Metamodell vom UML-Metamodell, in dem zwei verschiedene Namen auch immer zwei unterschiedliche Modellelemente kennzeichnen, wie zum Beispiel „java.awt.List“ und „java.util.List“.

In Abbildung 6.3 ist eine Situation dargestellt, die `Proxy` Elemente erfordert: In den Bereichen „Konstruktion“, „Produktion“ und „Produktion“ spielt das Teil (z.B. „Schraube“, „Motor“, „Platine“) als Konzept der Geschäftswelt eine Rolle, das als Bestandteil der Produktionsgüter des Unternehmens entwickelt, hinzugekauft und hergestellt werden muß. Dabei handelt es sich um ein und dasselbe konzeptuelle Element. Es muß deshalb in den drei Bereichen modelliert werden. Dies geschieht mithilfe der `Proxy` Elemente. Sollte in einem Bereich ein Element mit gleichen Namen sein, das aber ein anderes Konzept der Geschäftswelt modelliert, so kann für ein `Proxy` ein Aliasname vergeben werden.

Im Unterschied zu den Import-Beziehungen in UML, die andere Namensräume (Packages oder Klassen) in einem Namensraum einbinden, erlauben `Proxy` Elemente nicht nur eine Erweiterung des Namensraumes, sondern auch eine Möglichkeit, ein Modellelement in verschiedenen `Domains` zu bearbeiten. Das bedingt jedoch, daß es nur `Proxies` für `Classifier` Elemente gibt und keine für `Domains`.

Attribute

alias Dieses Attribut legt – wenn erforderlich – einen anderen Namen für ein durch ein Proxy importiertes Modellelement fest.

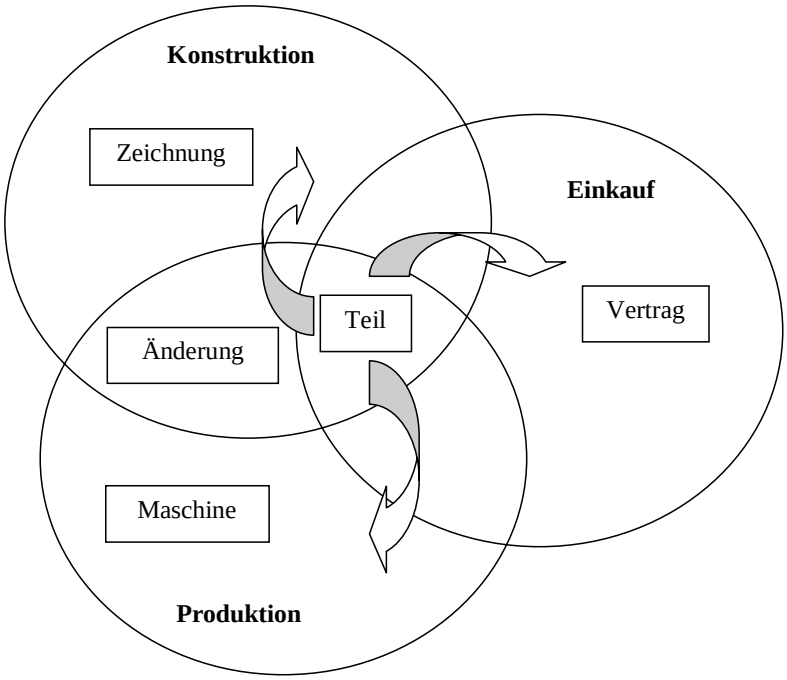

Abbildung 6.3: Keine strikte Trennung von Modellelementen zwischen Bereichen

Referenzen

master Verweist auf das Modellelement, daß in eine `Domain` eingeblendet werden soll.

Regel 8 (Kein Proxy für ein Element der gleichen Domäne) *Ein `Proxy` für ein Element der eigenen `Domain` ist nicht sinnvoll und deshalb nicht erlaubt.*

```
context Proxy
inv:
  self.namespace <>  self.master -> namespace
```

6.3.1.3 BusinessObject

Ein `BusinessObject` stellt die Generalisierung aller Konzepte der Geschäftswelt dar, die durch ein Business Object Modell modelliert werden sollen. `BusinessObject` ist im Metamodell eine abstrakte Klasse und gehört damit

nicht zu den Elementen, mit denen im Business Object Modell konkret modelliert wird. Generell dienen `BusinessObjects` und alle davon abgeleiteten Metamodell-Klassen der Klassifikation einer Menge von Objekten ihrer Modell-Instanzen, indem durch sie eine gemeinsame Struktur durch Attribute und Assoziationen und ein gemeinsames Verhalten durch Methoden und Events festgelegt wird.

Ein `BusinessObject` entspricht in dem oben genannten Sinne einer Klasse in einem objektorientierten Modell. Im Unterschied zu dem generischen Konzept einer Klasse, impliziert die Verwendung der konkreten Unterklassen von `BusinessObject` bestimmte Eigenschaften und Abhängigkeiten im Modell, die allein durch die Angabe der Attribute, Methoden, Assoziationen und Events nicht festgelegt sind. Diese implizite Semantik muß einerseits dem Modellierer bewußt sein, andererseits soll diese bei der Generierung eines lauffähigen Systems aus dem Modell gewährleistet sein. In den folgenden Abschnitten wird diese implizite Semantik der konkreten `BusinessObject` Unterklassen beschreiben, wobei eine Ergänzung dieser Semantik in der Beschreibung des Packages „Dynamics“ erfolgt.

6.3.1.4 BusinessEntityObject

Definition 6.1 (Business Entity Object (BEO)) *Ein Business Entity Object ist ein `Business Object` zur `Abbildung` eines passiven, zustandsbehafteten Konzeptes der `Geschäftswelt`. Passiv bedeutet, daß von einen BEO keine Manipulationen anderer `Business Objects` ausgehen. Der Zustand eines BEOs wird durch seine Attribute definiert. Durch die Operationen eines BEOs kann nur der Zustand des jeweiligen BEOs verändert werden.*

BEOs können mit anderen Business Objects durch verschiedene Relationen verknüpft werden:

- Generalisierungsrelation
- Namensraumrelation
- Nutzungsbeziehung
- Navigationsbeziehung
- Kompositionsbeziehung

- Ereignisbeziehung

Eine Generalisierungsrelation zwischen zwei BEOs bedeutet, daß das abgeleitete BEO den Inhalt des Namensraumes desjenigen BEOs übernimmt, von dem es abgeleitet wird. Das heißt, alle Attribute, Operationen und Relationen kommen zu den lokal im abgeleiteten BEO definierten hinzu. Im Business Object Metamodell ist dabei keine Namensgleichheit insbesondere für Operationen vorgesehen, ein Überschreiben oder „Overriding“ ist nicht möglich.

Navigationsbeziehung, Kompositionsbeziehung und Ereignisbeziehung werden im Abschnitt über das `Association` Modellelement beschrieben.

Regel 9 (Namensraumregel für BusinessEntityObjects) *Die Namensraumrelation* `Contains`*, die im Package „Base“ eingeführt wurde, wird für das BEO beschränkt auf die Elemente* `DataType`*,* `Reference`*,* `Operation`*,* `Attribute` *und* `Event`*. Ein BEO kann also insbesondere kein anderes BEO enthalten. Dies geschieht über die Kompositionsbeziehung.*

```
context BusinessEntityObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(DataType} or
    e.oclIsKindOf(Reference) or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )
```

6.3.1.5 BusinessServiceObject

Ein `BusinessServiceObject` – BSO – ist ein Element, das Manipulationen auf ein oder mehrere BEOs modelliert. Ein BSO hat einen Zustand, der bestimmt wird durch die Abfolge seiner Operationen, die von anderen Business Objects oder sich selbst aufgerufen wurden. Dieser Zustand kann entweder für einen Client des BSOs gehalten werden (Session-Semantik) oder für alle Clients (Service-Semantik). Im letzten Fall hat das Attribut `isSingleton` den Wert `true`.

Ein BSO stellt im Gegensatz zu den im folgenden vorgestellten `BusinessWorkflowObjects` keine länger dauernden Prozesse auf Business Objects dar.

Die vom BSO angebotenen Operationen sollen aus technischer Sicht transaktional, das heißt atomar, konsistent und dauerhaft, Zustände anderer Business Objects manipulieren bzw. Business Objects und ihre Abhängigkeiten untereinander erzeugen oder löschen.

Auch sind BSOs nicht wie die `BusinessTaskObjects` mit `BusinessActorObjects` verknüpft. Damit sind sie ähnlich wie BEOs passive Modellelemente, ihre Operationen werden immer von einem `BusinessTaskObject` angestoßen.

Obwohl eine direkte Abbildung für BSOs aus der Geschäftswelt schwierig anzugeben ist, kann man sich ein BSO in etwa als Handlungsanweisung oder Verfahrensbeschreibung vorstellen. Diese Anweisungen bzw. Beschreibungen sind in den seltensten Fällen als Dokument oder ähnliches in der Geschäftswelt vorhanden, sondern existieren nur in den Köpfen der Mitarbeiter eines Unternehmens. Die Operationen, die ein BSO realisiert, stellen diesen Teil der Geschäftslogik dar. Ein Beispiel für ein BSO mit Session-Semantik wäre „Stücklistenkalkulation“ mit der Operation „Stückliste auflösen()“, ein Beispiel für ein BSO mit Service-Semantik wäre „Unternehmenskalender“ mit der Operation „Arbeitstage(vonDatum, bisDatum)“.

Definition 6.2 (Business Service Object (BSO)) *Ein Business Service Object ist ein `Business Object` zur `Abbildung` von Handlungsanweisungen oder Verfahrensbeschreibungen der `Geschäftswelt`. Es stellt somit Dienste dar, die die Zustände bzw. die Abhängigkeiten anderer `Business Objects` analysieren oder verändern. Ein BSO stellt im Gegensatz zu den `Business Workflow Objects` keine länger dauernden Prozesse auf `Business Objects` dar. Die vom BSO angebotenen Operationen sollen aus technischer Sicht transaktional, das heißt atomar, konsistent und dauerhaft sein.*

Die möglichen Verbindungen zu anderen Business Objects werden im Abschnitt zum Modellelement `Association` beschrieben.

Attribute

isSingleton Dieses Attribut legt die Semantik fest, die für den Zustand des BSOs aus Sicht der aufrufenden Business Objects gilt: Hat `isSingleton` den Wert `true` so gibt es (in Ebene M1) nur eine Instanz eines BSOs und dessen Zustand gilt für alle aufrufenden Business Objects. Im anderen Fall gibt es pro aufrufenden Business Object je eine Instanz und damit Zustand des BEOs.

Regel 10 (Namensraumregel für BusinessServiceObjects) *Ein BSO kann folgende Elemente enthalten* `DataType`*,* `Operation`*,* `Attribute` *und* `Event`*.*

```
context BusinessServiceObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(DataType} or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )
```

6.3.1.6 Association

Elemente der Geschäftswelt existieren nicht unabhängig voneinander. Viele Elemente in Verbindung mit anderen Elementen, wie zum Beispiel ein „Kunde“ in Verbindung steht mit ein oder mehreren „Aufträgen“.

Um solche Verbindungen im Business Object Modell modellieren zu können, gibt es im Metamodell zwei Möglichkeiten:

- `Association`: Verbindung zwischen zwei `BusinessObjects` mit wählbarer Multiplizität, Navigationsmöglichkeit und Aggregationsart.
- `Attribut`: Verbindung zwischen zwei `BusinessObjects` mit festgelegten Parametern.

Eine `Association` zwischen zwei `BusinessObjects` erfolgt mithilfe der nachfolgend beschriebenen `AssociationEnds`. Eine `Association` enthält jeweils zwei `AssociationEnds`. Diese bestimmen auch die jeweilige Multiplizität, Navigationsmöglichkeit und Aggregationsart.

Regel 11 (Modellelemente einer Association) *In einer* `Association` *kann nur ein Modellelement enthalten sein, und zwar das* `AssociationEnd` *Element.*

```
context Association
inv:
  self.contents.forAll( e |
                       e.oclIsKindOf(AssociationEnd) )
```

Regel 12 (Keine Generalisierungsrelation für Associations) *Eine Generalisierungsrelation zwischen zwei* `Associations` *ist nicht erlaubt, da eine* `Association` *weder Attribute noch Operationen enthält, die vererbt werden könnten.*

```
context Association
inv:
  self.supertypes -> isEmpty and
  self.isRoot and
  self.isLeaf and
  not self.isAbstract
```

Regel 13 (Associations sind binär) *Eine* `Association` *verbindet immer genau zwei* `BusinessObjects`*.*

```
context Association
inv:
  self.contents ->
    select(a | a.oclIsTypeOf(AssociationEnd) )
      -> size = 2
```

6.3.1.7 AssociationEnd

Eine `Association` enthält immer zwei `AssociationEnds`. Die Aufgabe dieser `AssociationEnds` ist es, die `BusinessObject` Klassen zu bestimmen, zwischen denen die `Association` besteht, als auch die Verknüpfungseigenschaften festzulegen.

Die Verknüpfungseigenschaften werden mithilfe der Attribute der Klasse `AssociationEnds` festgelegt. Zu den Eigenschaften gehören:

- Aggregationsart
- Multiplizität
- Zugriffsrechte
- Navigationsmöglichkeit

Attribute

aggregation Dieses Attribut legt die Art fest, in welcher Weise der Lebenszyklus der miteinander verbundenen `BusinessObjects` durch die `Association` beeinflußt wird. Es gibt dabei drei verschiedene Aggregationsarten:

- „independent“: Die Lebenszyklen der miteinander verbundenen `BusinessObjects` sind vollkommen unabhängig voneinander. So existieren Instanzen der `BusinessEntityObjects` „Kunde“ und „Berater“ unabhängig voneinander, auch wenn zwischen ihnen bestehende Instanzen der `Association` „betreut/wird betreut von“ geändert oder gelöscht werden.
- „aggregate“: Definiert eine Aggregation zwischen zwei `BusinessObjects`. Dabei sind die Instanzen der eine Seite der Verknüpfung Teil einer Instanz auf der anderen Seite. Im Gegensatz zu der nachfolgend beschriebenen Variante der Komposition können in einer Aggregation Instanzen Teil mehrerer anderer Instanzen sein. Die Lebenszyklen der miteinander verbunden Instanzen ist nicht unabhängig. Zu einem Teil-Element muß mindestens immer ein Element existieren, das dieses enthält. So bildet die `Association` „besitzt/Eigentum von“ zwischen den `BusinessEntityObjects` „Gesellschaft“ und „Flugzeug“ eine Aggregation. Ein Flugzeug kann von mehreren Fluggesellschaften geteilt werden. Ein „herrenloses“ Flugzeug ist aber nicht möglich.
- „composite“: Eine Komposition ist eine striktere Form einer Aggregation. Zusätzlich zur Aggregation wird gefordert, daß in einer Komposition Instanzen nur Teil *genau einer* anderen Instanz sein dürfen. Das hat Auswirkungen auf den Lebenszyklus der Teil-Elemente: Wird die Instanz gelöscht, dessen Teil sie sind, werden auch sie gelöscht. Beispiel für eine Komposition wäre die `Association` „besteht aus/Teil von“ zwischen den `BusinessEntityObjects` „Raum“ und „Fenster“.

multiplicity Legt fest, wieviel Instanzen des mit dem `AssociationEnd` bestimmten `BusinessObject` an die `Association` geknüpft werden können. Der Wert dieses Attributs ist ein Bereich $[l, u]$, wobei $l, u \in \mathrm{N}$ und $l \leq u \wedge l \geq 0 \wedge u \neq 0$.

isNavigable Bestimmt, ob von einem Ende der `Association` zu diesem `AssociationEnd` zugegriffen werden kann. Mit diesem Attribut kann

die Navigationsmöglichkeiten zwischen Instanzen in einer `Association` eingeschränkt werden.

Regel 14 (Nur Associations zwischen BusinessObjects) *:*

```
context AssociationEnd
inv:
  self.type.oclIsTypeOf(BusinessObject)
```

Regel 15 (Aggregationen nur in einer Richtung) *:*

```
context AssociationEnd
inv:
  self.aggregation <> #independent
    implies (self.container.contents ->
          select(a | a.oclIsKindOf(AssociationEnd)
                   and a <> self)) = #independent
    select(a | a.oclIsTypeOf(AssociationEnd) )
                                        -> size = 2
```

6.3.1.8 Attribut

Jeder `Classifier` kann als strukturelle Eigenschaft in seiner Containment-Relation eine Menge `Attribute` enthalten, die auch leer sein kann. Der Typ eines `Attributs` wird durch einen `DataType` bestimmt. Da ein `DataType` auch ein `Classifier` sein kann, steht, wie oben bereits erwähnt, ein Attribut für eine Verbindung zwischen zwei Klassen. Im Gegensatz zu einer `Association` kann eine Verbindung über ein Attribut keine Aggregation oder Komposition sein, sowie auch keine Multiplizität abweichend von $[0, 1]$ besitzen.

Als besondere Eigenschaft kann ein `Attribut` – ebenso wie eine `Operation` – entweder auf Instanz oder Klassenebene gelten. Gilt das `Attribut` auf Klassenebene, so wird der Wert des `Attributs` von allen Instanzen geteilt. Diese Eigenschaft wird als Attribut `scope` von der Klasse `Feature` geerbt.

6.3.2 Dynamik

Ergänzend zu den strukturellen Aspekten werden in diesem Abschnitt die Modellelemente eingeführt, die zur Klassifizierung verschiedener Elemente der Geschäftswelt dienen, die eine Änderung der Struktur dienen. Diese Aspekte werden im Package „Dynamics“ zusammengefaßt.

Mit dem Business Object Modell sollen folgende Aspekte ausgedrückt werden können:

- Manipulationen auf Business Objects
- Benachrichtigungen zwischen Business Objects
- Kontrollfluß zwischen Business Activity Objects innerhalb eines Workflows (wird in Abschnitt 6.4 eingeführt)

Alle Modellelemente des Packages „Dynamics“ werden in Abbildung 6.4 dargestellt. Dabei sind Klassen aus anderen Packages „Base“ und „Structure“ grau dargestellt. In den folgenden Abschnitten werden die Modellelemente diskutiert.

6.3.2.1 Operation

das Element `Method`, das im nachfolgenden Abschnitt beschrieben wird, ermöglicht eine Manipulation des Business Object Systems. Die Schnittstelle einer solchen Manipulation stellt das Element `Operation` dar. Die Funktion einer `Operation` entspricht die der Methodendeklaration in traditionellen objektorientierten Systemen. `Operationen` spezifizieren dabei lediglich die Namen und Signaturen einer `Method`.

Die Spezifikation erfolgt mithilfe der später vorgestellten `Parameter` Elemente. Eine `Operation` mit Parametern enthält dazu in ihrem Namensraum eine Anzahl von `Parameter` Elementen.

Im Business Object Metamodell gibt es eine Reihe „fest verdrahteter“ Operationen.:

- Business Object: `create(), delete(), subscribe(), unsubscribe(), registerFor(), deregister()`

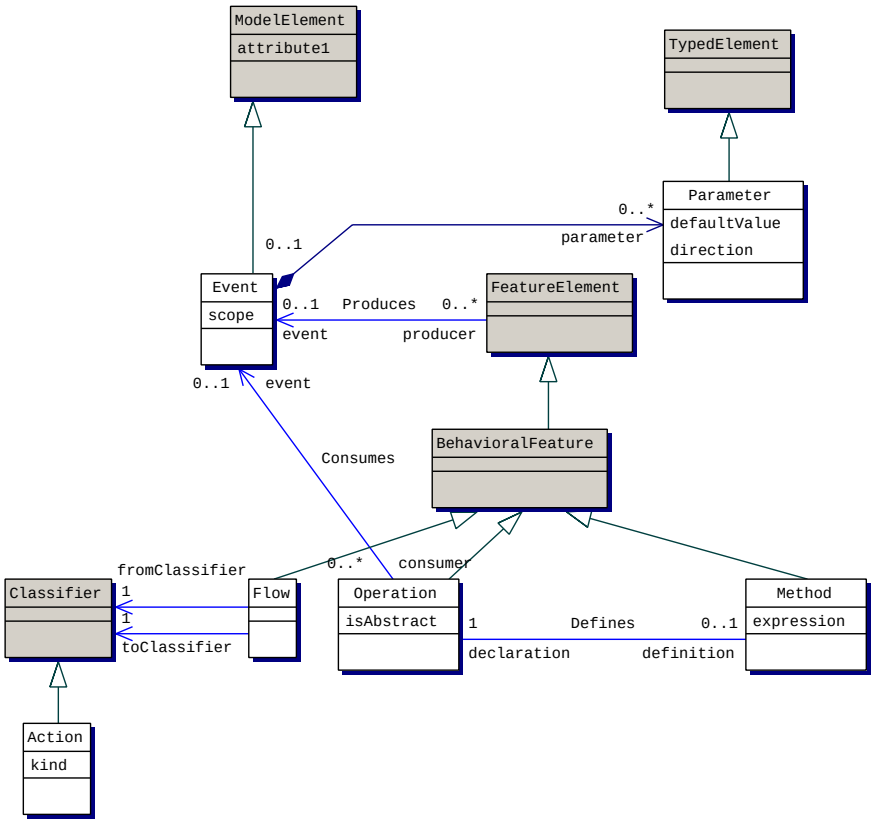

Abbildung 6.4: Verhaltenselemente des Business Object Metamodells (Package `Dynamics`)

- Business Activity Object: `resume(), terminate(), abort(), suspend(), complete()`
- Business Workflow Object: `resume(), terminate(), abort(), suspend(), start()`

Diese `Operationen` werden in Ebene M1 eines Business Object Modells als Standardoperationen von Business Objects deklariert und müssen von jedem Business Object System implementiert werden.

Ist jedoch nicht möglich, solche Framework-Operationen direkt in einem Metamodell zu definieren, das – wie die MOF – eine strikte Trennung zwischen den Modellebenen vorsieht. Aus diesem Grund können diese Operationen hier nicht zusammen mit den Ebene M2 Elementen definiert werden. Sie werden später im Kapitel 9 näher beschrieben, in dem eine Abbildung des Business Object Modells eine Infrastruktur-Framework besprochen wird.

Ebenso wie Attribute können `Operationen` auf Instanz- oder Klassenebene deklariert werden. Gilt die `Operation` auf Klassenebene, so können mit der `Methode` alle `Attribute` auf Klassenebene manipuliert werden. Die `Methoden` von `Operationen` auf Instanzebene können alle `Attribute` manipulieren. Diese Eigenschaft wird als Attribut `scope` von der Klasse `Feature` geerbt.

Attribute

isAbstract Dieses Attribut legt fest, ob es eine Implementierung zur Deklaration der `Operation` geben darf. Ist dieses Attribut „true“ gibt es auf Ebene M1 des Business Object Modells keine Realisierung der `Operation`.

Referenzen

definition Mit der Referenz `Defines` werden die `Methoden` ihren zugehörigen `Operationen` zugeordnet. Ist die `Operation` als abstrakt deklariert, ist `definition` nicht belegt.

event Im Event-Konzept des Business Object Metamodells gibt jeweils Event-Produzenten (Producer) und Event-Verarbeiter (Consumer). Ein Producer kann jedes `Feature` des Business Objects Metamodells sein, also `Methoden`, `Attribute`, und `Flows`. Verarbeitet können solche Events jedoch nur in `Methoden`. Das wird in der Deklaration der `Methode`, der

`Operation`, angegeben. Diese werden durch die Referenz `Consumes` ausgedrückt. Im Business Object Metamodell ist die Zahl der Events, die eine Methode verarbeiten kann auf eins begrenzt.

Regel 16 (Namensraumregel für Operationen) *Die Namensraumrelation* `Contains`*, die im Package „Base“ eingeführt wurde, wird für eine Operation beschränkt auf das Element* `Parameter`*.*

```
context Operation
inv:
  self.contents.forAll( e | e.oclIsKindOf(Parameter} )
```

6.3.2.2 Method

Das `Method`-Element steht für eine ausführbare Definition einer `Operation`. Dabei wird im Business Object Metamodell nicht weiter aufgeteilt in Anweisungen, Ausdrücke oder Schleifen. Es ist nur ein Attribut `expression` vorgesehen, für dessen textuellen Inhalt im Metamodell keine Einschränkungen oder Regeln vorgesehen sind. Um das Metamodell flexibel und unabhängig von Beziehungen zu einer speziellen Programmiersprache zu halten, wird stattdessen die Entscheidung, was eine `expression` ist und welche Sprachelemente beziehungsweise Framework-Komponenten sie enthalten darf, erst in der Abbildung des Metamodells auf eine Business Object Infrastruktur definiert. In dieser Arbeit erfolgt die Abbildung auf eine eingeschränkte Java-Variante. Welche Sprachelemente genutzt werden, wird im Kapitel 9 beschrieben.

Attribute

expression Ein Platzhalter für in einer Business Object Infrastruktur ausführbare Methodenrümpfe. Einschränkungen für den Inhalt dieses Attributs erfolgen in der Beschreibung der Abbildung auf die jeweilige Infrastruktur.

Referenzen

declaration Referenz auf die zugehörige `Operation`. Jede `Methode` hat genau eine `declaration`.

6.3.2.3 Parameter

Ein `Parameter`-Element ist enthalten im Namensraum einer `Operation`. Dabei wird unterschieden zwischen verschiedenen Parametertypen:

- in: Für Parameter, die einer Operation übergeben werden.
- out: Für Parameter, die als Ausgabe einer Operation dienen.
- in/out: Parameter, die sowohl als Ein- als auch als Ausgabewerte für Operationen dienen.
- return: Ein spezieller Parameter, der den Rückgabewert einer Operation definiert. Pro Operation ist genau ein Return-Parameter erforderlich.

Attribute

direction Bestimmt den Parametertyp. Mögliche Werte sind – wie oben bereits beschrieben – „in“, „out“, „inout“ und „return“.

defaultValue Mit `defaultValue` läßt sich für Parameter der Typen in und in/out ein Standardwert festlegen.

6.3.2.4 Event

In einem Business Object System sind starre Verknüpfungen (z.B. mithilfe des `Association`-Elements) zwischen Business Objects nicht immer wünschenswert oder gar nicht modellierbar.

Nicht modellierbar, da nicht alle denkbaren Verbindungen zum Zeitpunkt einer Spezifikation bekannt sind. So könnte ein Business Object System mit einem Business Object „Kunde“ erweitert werden um ein CRM-Modul. In diesem sollen alle Zugriffe des Kunden auf ein zugriffsbeschränktes Internet-Angebot im Business Object „Sitzung“ gespeichert werden. Da die Verbindung zwischen „Kunde“ und „Sitzung“ zur Spezifikationszeit des Business Objects „Kunde“ noch nicht bekannt war, muß dieses geändert werden.

Starre Verknüpfungen sind nicht wünschenswert, da eine zu starre Verbindung zwischen verschiedenen Business Objects die Wiederverwendung in anderen Kontexten erschwert oder sogar verhindert. So kann ein Business Object „Teil“

eines PDM-Systems, das starr mit einem Business Object „Version“ verknüpft ist, nur dann im Kontext eines PPM-Systems wiederverwendet werden, wenn auch das eigentlich im PPM-Kontext sinnlose Business Object „Version“ wiederverwendet wird.

Um diese Nachteile einer starren Koppelung zu umgehen, bietet das Business Object Metamodell die Möglichkeit zur losen Koppelung unter Business Objects mithilfe eines Event-Mechanismus. Die Idee dabei ist, das jedes `Feature` (Attribut, Operation, Flow) eines Business Objects einen Event auslösen kann. Zum Beispiel, wenn eine Operation aufgerufen wurde, oder sich der Wert eines Attributs geändert hat.

Die Realisierung dieses Event-Konzeptes im Metamodell folgt dem Publisher/-Subscriber-Pattern nach [GHJV94]. Es gibt Event Producer und einem Event Consumer. Dies sind die Operationen der Business Objects. Dabei ist es möglich, Events auf Instanzebene, oder auf Ebene der Klassen zu definieren.

- Auf Ebene der Klassen: Jedes Mal, wenn ein neues Business Object X erzeugt wird, erhält der Abonnent dieses Events eine Nachricht.
- Auf Ebene der Instanzen: Eine Instanz eines Business Workflow Objects „Genehmigungsverfahren“ wartet auf die Beendigung der Aktivität „Dokument prüfen“ (Instanz eines Business Activity Objects).

Der Eventmechanismus eignet sich neben der losen Koppelung von Business Objects auch zu einer generischen, Infrastruktur-unabhängigen Implementierung einer einfachen Workflow- Funktionalität. Auf diesen Aspekt wird im folgenden Abschnitt näher eingegangen.

Das `Event`-Element bildet den Mittelpunkt des Event-Mechanismus. Jeder Event-Producer produziert eine `Event`-Instanz, die von einer `Operation` als Event-Consumer verarbeitet wird. Die (optionalen) Parameter der `Event`-Elements werden dabei an die `Operation` übergeben.

Attribute

scope Mit `scope` wird festgelegt, ob ein `Event` auf Klassenebene („class“) oder Instanzebene („instance“) generiert werden soll.

Referenzen

consumer Referenz auf die zugehörige `Operation`, die ausgeführt werden soll, wenn der Event auftritt.

producer Verweis auf das Event-verursachende `Feature`.

parameter Optionale Liste an `Parametern` des `Events`

6.4 Integration von Workflows

Die in den folgenden Abschnitten vorgestellten Modellelemente `BusinessWorkflowObject` (BWO), `BusinessTaskObject` (BTO) und `BusinessActorObject` (BAO) dienen hauptsächlich der Integration einer Modellierung arbeitsteiliger Vorgänge in das Business Object Metamodell. Arbeitsteilige Vorgänge sind diejenigen, meist längerdauernden Prozesse, in denen mehrere Akteure nach einer festgelegten Folge von einzelnen, jeweils von einem Akteur bearbeitetet Aufgaben Business Objects erzeugen, manipulieren oder löschen. Darüberhinaus können mit den Modellelementen auch nicht-arbeitsteilige Vorgänge, das heißt Vorgänge die von weniger als zwei Akteuren bearbeitet werden oder die nur eine Aufgabe enthalten, ausgedrückt werden.

6.4.1 Relevante Aspekte

Im Abschnitt 4.1 wurden verschiedene Aspekte arbeitsteiliger Vorgänge dargestellt. Diese Aspekte drücken sich in den Metamodellen von Workflow-Management-Systemen – speziell in den dafür vorgesehenen Definitionswerkzeugen für Workflow-Schema – durch bestimmte Modellelemente aus. Im Fall des Business Object Metamodells werden jedoch nicht alle diese Modellelemente übernommen, da einige Aspekte für ein Business Object Modell nicht relevant sind, beziehungsweise andere Modellelemente erfordern als in einem traditionellen Workflow-Metamodell. Im folgenden werden die Aspekte nach ihrer Relevanz für das Business Object Metamodell betrachtet.

6.4.1.1 Funktionaler Aspekt

Der funktionale Aspekt eines Workflow-Modells beschreibt, welche Aufgaben im Verlauf der Ausführung eines Vorgangs überhaupt zu bearbeiten sind. Zur Darstellung des funktionalen Aspektes sind Modellelemente zur Verfügung zu stellen,

die eine Definition von Aufgaben sowie deren Gliederung umfaßt. Das in vielen Workflow-Metamodellen [Sch99, JBS97, LR00] gewählte hierarchische Schema – ein kompositionales Workflow-Element, das sich (Subworkflows) und die Elemente zur Darstellung der Arbeitsaufgaben (Tasks) enthalten kann – wird auch im Business Object Metamodell mit zwei korrespondierenden Elementen zur Verfügung gestellt:

Zum einen das `BusinessWorkflowObject` (BWO) als Strukturmittel zur Beschreibung der Vorgänge an sich und zum anderen das `BusinessTaskObject` zur Beschreibung der in den BWOs zu erledigenden Aufgaben. Sie sind mithilfe der `Contains` Beziehung des Modellelements `Namespace` in den Namensraum eines BWOs eingebettet. Zusätzlich zu BTOs können BWOs auch andere BWOs enthalten, das heißt, hierarchische Vorgangsdefinitionen sind möglich. Außerdem enthalten BWOs die `Action` Modellelemente, die zur Steuerung des Kontrollflusses dienen. Sie werden im folgenden Abschnitt beschrieben, der den Verhaltensaspekt eines Workflow-Modells diskutiert.

6.4.1.2 Verhaltensaspekt

Der Verhaltensaspekt beschreibt die kausalen Zusammenhänge zwischen BTOs und (Sub-)BWOs in einem BWO. Das heißt, welches ist der erste Schritt in einem Workflow, welche Schritte kommen nach einem Schritt und nach welchem Schritt ist der Workflow abgeschlossen. Dazu werden Modellierungsmöglichkeiten gebraucht, die den Kontrollfluß innerhalb eines Vorgangs festlegen. Im Business Object Metamodell werden nur existentielle, keine qualitativen Aussagen über einen kausalen Zusammenhang zwischen zwei Modellelementen eines BWOs getroffen. Es werden zu diesem Zweck die im Package „Dynamics“ beschriebenen Modellelemente `Flow` und `Action` eingeführt.

`Flow` ist eine gerichtete Verbindung zwischen zwei Elementen, die in einem BWO enthalten sein dürfen, also BWO, BTO und `Action`. Es modelliert dabei folgenden Zusammenhang: Ist die Bearbeitung einer Aufgabe, die durch eine der drei genannten Modellelemente dargestellt wird, abgeschlossen, so wird diejenige Aufgabe benachrichtigt, in der die `Flow` Verbindung zeigt.

Eine `Flow` Verbindung nur zwischen BTOs bzw. BWOs würde nur die Modellierung sequentieller und paralleler Abläufe ermöglichen, eine Fallunterscheidung oder ein Zusammenführen mehrerer paralleler Abläufe ist damit noch nicht möglich. Außerdem wäre noch nicht festgelegt, mit welcher Aufgabe der Vorgang starten soll und mit der Beendigung welcher Aufgabe der Vorgang beendet wird.

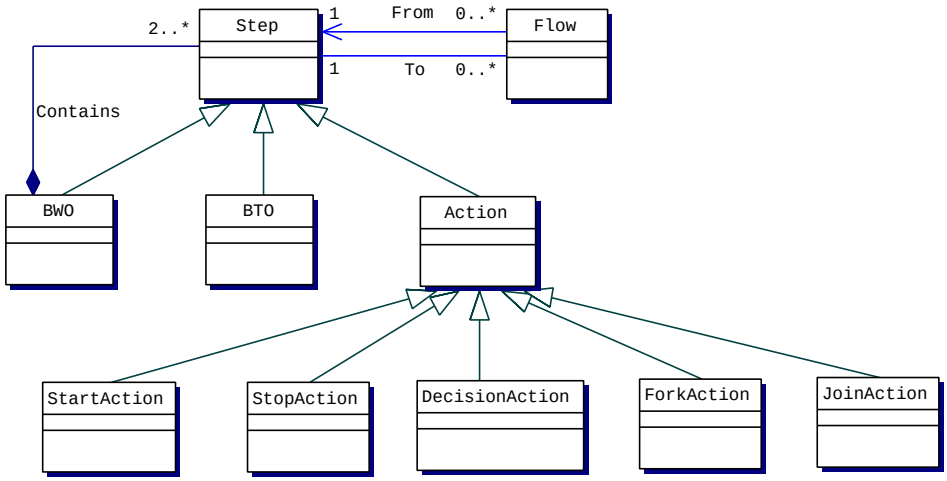

Abbildung 6.5: Abstraktes Metamodell des funktionalen Aspektes

Zur Modellierung dieser Eigenschaften werden die `Action` Elemente eingeführt. Sie kennzeichnen Start- und Endpunkte von Vorgängen und ermöglichen, bedingte Abläufe festzulegen.

Abbildung 6.5 zeigt die abstrakten Zusammenhänge zwischen den Modellelementen. Dabei ist zu beachten, daß dies kein Ausschnitt des konkreten Metamodells ist. Außerdem werden bestimmte Eigenschaften in der Abbildung nicht ausgedrückt, wie zum Beispiel, daß ein Decision Element mit drei (verschiedenen) Flow Elementen verknüpft sein muß und zwar einmal in einer „to“ Beziehung und zweimal in einer „from“ Beziehung. Diese Eigenschaften werden später in den Regeln des Packages „Dynamics“ festgelegt.

6.4.1.3 Informationsbezogener Aspekt

In jedem Teilschritt eines Vorganges werden Daten erzeugt, konsumiert, manipuliert oder gelöscht. Diesen Aspekt wird informationsbezogener Aspekt genannt. In traditionellen Workflowmodellen ist dieser Aspekt noch stark von einem strukturalem Paradigma geprägt. Danach werden einem Prozeßschritt vor dessen Ausführung Daten zur Verfügung gestellt. Im Prozeßschritt werden die Daten manipuliert und nach Abschluß des Schritts einem nachfolgenden Schritt als Eingabe zur Verfügung gestellt. Dabei können diese Daten einfache Variablen einfacher Datentypen sein, wie auch komplette Dokumente, die zur Bearbeitung von einem Prozeßschritt zum nächsten gereicht werden. Oftmals wird zudem der Kontrollfluß dadurch gesteuert, daß ein Prozeßschritt erst dann ausgeführt werden kann,

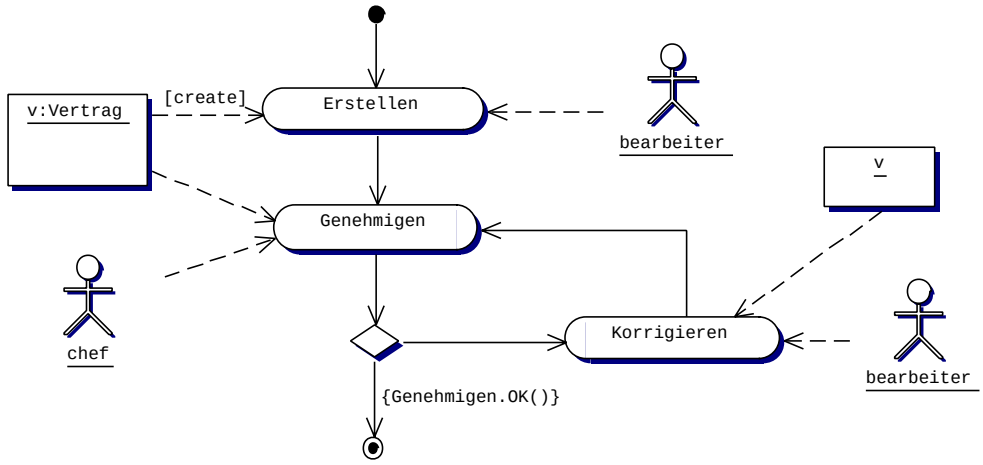

Abbildung 6.6: Beispiel für die Darstellung des informationsbezogenen Aspektes in einem UML-Aktivitätsdiagramm

wenn alle Eingangdaten, die für diesen Schritt festgelegt sind, zur Verfügung stehen.

In einem Business Object Metamodell, das auf der Modellierung von Konzepten der Geschäftswelt und ihrer Beziehung untereinander basiert, paßt ein solcher Ansatz nicht, da es keine nicht-identifizierbare Menge an „Daten“ bzw. freie Menge an (prozeßglobalen) Variablen gibt. Stattdessen ist im Business Object Metamodell jede Aufgabe mit ein oder mehreren BEOs oder BSOs verknüpft. Die Verknüpfung wird durch das Modellelement `Workproduct` dargestellt, das im Abschnitt über das `Association` Element vorgestellt wird.

Konkret bedeutet dies, daß innerhalb einer Aufgabe nur auf die vorher mit der Verknüpfung `Workproduct` assoziierten Business Objects zugegriffen werden kann. Um nicht nur den Zugriff auf eine ganze Klasse von Business Objects zu modellieren, sondern auf eine konkrete Instanz, werden `Object` Modellelemente eingeführt, die (auf Ebene M1) eine Instanz eines Business Objects darstellen. Durch die Angabe von Namen für diese Instanzen, können so in einem Vorgang mehrere Aufgaben eine bestimmte Business Object Instanz manipulieren.

In Abbildung 6.6 ist eine Darstellung eines Vorgangs zur Erstellung eines Vertrages abgebildet. Der Vorgang umfaßt die BTOs „Erstellen“, „Genehmigen“ und „Korrigieren“. Manipuliert wird in allen drei Aufgaben eine Instanz des BEOs „Vertrag“, hier mit „v“ bezeichnet. Die `Workproduct` Beziehungen sind durch gerichtete, gestrichelte Linien vom BEO (als Objekt dargestellt) zum BTO visualisiert. Für die Aufgabe „Erstellen“ ist die Beziehung um das Tag „[crea-

te]“ ergänzt, um zu modellieren, daß zunächst eine Instanz eines BEOs „Vertrag“ erzeugt werden muß. Diese Instanz ist dann an den Namen „v“ gebunden und kann – wie für das BTO „Korrigieren“ dargestellt – durch ein anderes Objektsymbol wiederverwendet werden. Eine genaue Darstellung der Nutzung von UML Aktivitätsdiagrammen zur Darstellung von BWOs erfolgt im Kapitel 7.6.

6.4.1.4 Operationaler Aspekt

Der operationale Aspekt beschreibt in Workflow-Modellen, welche Hilfsmittel – Werkzeuge oder Arbeitsmittel – zur Ausführung einer Aufgabe benötigt bzw. aufgerufen werden soll. In den meisten Fällen wird in den Aufgaben spezifiziert, daß ein bestimmter Editor mit einem Dokument aufgerufen wird, oder eine Maske eines Hostsystems. Nicht zu unrecht wird dieser Aspekt auch teilweise technologischer Aspekt bezeichnet [Jab95], denn es ist eine Verknüpfung logischer Workflow-Aufgaben mit technischen Realisierungen.

Nach den Kriterien in Abschnitt 3.5 ist klar, das eine Verknüpfung eines Elements des Business Object Modells mit einem technischen Artefakt außerhalb der Geschäftswelt nicht vom Business Object selbst ausgehen darf. Deshalb ist im Business Object Metamodell kein Element zur Modellierung eines solchen Aspektes vorgesehen.

Natürlich ist ein Workflow, ohne die Möglichkeit, die in den Aufgaben zu manipulierenden Business Objects von außen durch den jeweiligen Bearbeiter zu bearbeiten, nur vorstellbar bei vollständig automatisierten Aufgaben. In der Regel müssen jedoch die (menschlichen) Bearbeiter von Aufgaben in einem Workflow die Business Objects interaktiv mithilfe von Werkzeugen bearbeiten.

Dazu wird im Business Object Modell das Eventkonzept genutzt. In Abbildung 6.7 ist ein beispielhafter Ablauf aus dem oben bereits vorgestellten Vorgang „Neuen Vertrag erstellen“ abgebildet. Die Grenze des eigentlichen Business Object Systems befindet sich dabei zwischen dem BTO „Erstellen“ und dem Client „Workplace“ eines Benutzers. Der Client „Workplace“ registriert sich beim BTO für den Event „Activation“. Wird das BTO aktiviert, dann wird der Client benachrichtigt. In diesem Fall ruft er einen Vertragseditor auf, mit dessen Hilfe der Bearbeiter der Aufgabe die entsprechende Instanz des BEOs „Vertrag“ manipulieren kann.

Mit diesem Mechanismus bleibt das Business Object „Erstellen“ frei von expliziten Verbindungen zu einem Werkzeug, das eine Sicht auf das Business Object Modell bereitstellt, jedoch kein Teil dieses Modells ist.

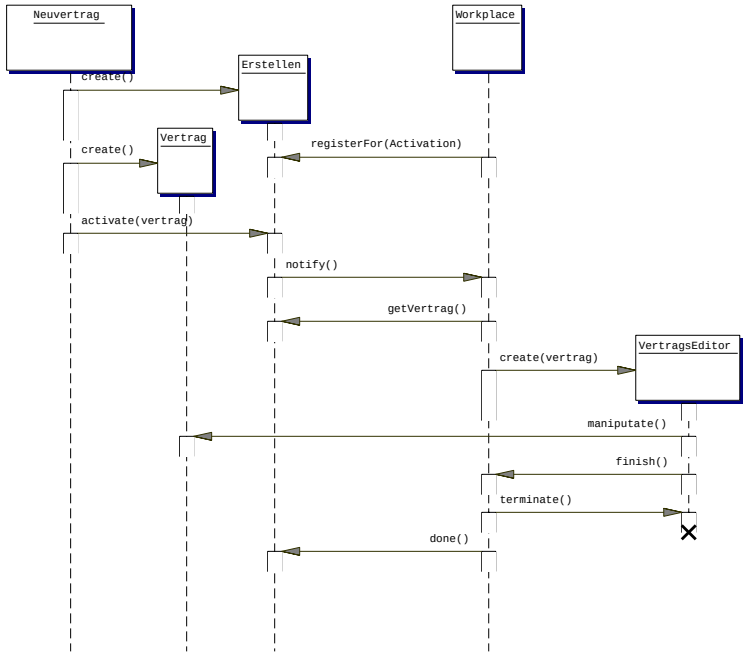

Abbildung 6.7: Beispiel einer Anbindung von Werkzeugen durch Events (UML-Sequenzdiagramm)

6.4.1.5 Organisationsaspekt

Der Organisationsaspekt beschreibt einerseits die Zuordnung von Rollen zu Aufgaben. Andererseits werden zur Laufzeit eines Vorgangs die konkreten Bearbeiter mit der entsprechenden Rolle und über zusätzlichen Eigenschaften bestimmt.

Mit den unterschiedlichen Rollen sind in der Regel organisatorische Strukturen verknüpft, die Betriebsmittel und Mitarbeiter zu gruppieren. Die Strukturen werden dazu genutzt neben der Rolle noch weitere Eingrenzungen für eine Auswahl von Bearbeitern einer Aufgabe zu machen. So soll eine Aufgabe „Kreditwürdigkeit prüfen“ zum Beispiel von der Rolle „Sachbearbeiter“ der Abteilung „Kreditvergabe“ erledigt werden. Die Strukturen sind dabei von der Art der Unternehmung oder Organisation abhängig und können nicht generell als Schema angegeben werden. In vielen Unternehmen spielen Abteilungen, Gruppen oder Bereiche eine Rolle, in einer Universität wird aber zum Beispiel nach Fakultäten, Instituten und Lehrstühlen gruppiert.

Im Business Object Metamodell werden zur Modellierung der Bearbeiter einer Aufgabe die `BusinessActorObjects` (BAOs) eingeführt. Mit ihrer Hilfe werden aktive Elemente der Geschäftswelt modelliert. Also Akteure, von denen

eine Initiative und Möglichkeit zur Änderung eines Zustands der Geschäftswelt ausgeht. Dies sind in der Regel Personen die für eine Organisation tätig sind, es können aber auch computerunterstützte, automatisierte Systeme sein.

Die BAOs sind im Business Object Metamodell Modellelement zur expliziten Modellierung der Akteure, das mit anderen Business Object in Beziehung stehen kann. Im Gegensatz dazu werden in traditionellen Informationssystemen Akteure nur im Rahmen der Authentifizierung und Autorisierung getrennt vom eigentlichen System modelliert.

Eine Zuweisung von Rollen, Bildung von organisatorischen Strukturen oder die Zuweisung von bestimmten Kompetenzen erfolgt über die Beziehung `Association` zwischen BAOs und BEOs. Das heißt, Rollen, Organisationsschemata oder sonstige Eigenschaften werden mithilfe von BEOs modelliert und den BAOs zugewiesen.

6.4.2 Struktur

Im folgenden werden die für die Integration von arbeitsteiligen Vorgängen notwendigen strukturellen Modellelemente des Metamodells vorgestellt. Sie ergänzen das im Abschnitt 6.3.1 beschriebene Package „Structure“. Eine Beschreibung der Modellelemente die die Ablauforganisation solcher Vorgänge beschreiben erfolgt in der Darstellung des Packages „Dynamics“ im Abschnitt 6.4.3.

6.4.2.1 BusinessWorkflowObject

Das `BusinessWorkflowObject` (BWO) ist eine Abstraktion für alle arbeitsteiligen Vorgänge. Es ist ein Strukturelement zur Gliederung von Vorgängen: Es kann sowohl einen eigenständigen Vorgang darstellen, wenn es nicht in ein anderes BWO eingebettet ist, als auch einen Teilvorgang beschreiben, wenn es Teil eines anderen BWOs ist. Die Schachtelungstiefe dieser hierarchischen Gliederung ist dabei nicht begrenzt.

Ein BWO bildet einen Namensraum nicht nur für Teilvorgänge, sondern auch für die Vorgangsschritte, die `BusinessTaskObjects` (BTOs), die im nächsten Abschnitt beschrieben werden. Die BWOs enthalten ferner `Action` und `Flow` Modellelemente, die den Kontrollfluß zwischen den Arbeitsschritten bzw. Teilvorgängen beschreiben. Zudem werden innerhalb eines BWOs den einzelnen BTOs Eigenschaften von `BusinessActorObjects` (BAOs) zugewiesen.

Definition 6.3 (Business Workflow Object (BWO)) *Ein Business Workflow Object (BWO) ist ein* `Business Object` *zur* `Abbildung` *von* `Geschäftsprozessen` *der* `Geschäftswelt`*. Ein BWO setzt sich zusammen aus weiteren BWOs (Subworkflows), den einzelnen, nicht weiter unterteilbaren Schritten (*`Business Task Objects` *und Elementen zur Bestimmung des Ablaufs des BWOs.*

Der Lebenszyklus eines BWOs ist bestimmt durch die Framework-Operationen, die dem BWO im Business Object Metamodell zugewiesen sind: `resume()`, `terminate()`, `abort()`, `suspend()` und `start()`. Diese Ebene M1 Operationen werden im Abschnitt über das Package „Operation“ näher beschrieben.

Regel 17 (Namensraumregel für BusinessWorkflowObjects) *Ein BWO kann folgende Elemente enthalten* `BusinessWorkflowObjects`*,* `BusinessTaskObjects`*,* `BusinessActorObjects`*,* `Flow`*,* `Action`*,* `Operation`*,* `Attribute` *und* `Event`*.*

```
context BusinessWorkflowObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(BusinessWorkflowObject) or
    e.oclIsKindOf(BusinessTaskObjects) or
    e.oclIsKindOf(BusinessActorObjects) or
    e.oclIsKindOf(Flow) or
    e.oclIsKindOf(Action) or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )
```

Regel 18 (Keine Generalisierungsrelation für BusinessWorkflowObjects) *Eine Generalisierungsbeziehung zwischen BWOs wird in dem in dieser Arbeit vorgestellen Business Object Metamodell nicht gestattet, da sie von der Bedeutung nicht der Generalisierungsbeziehung zwischen BEOs oder BSOs entsprechen würde. Während eine Generalisierungsbeziehung zwischen BEOs und BSOs eine Erweiterung bzw. Verfeinerung durch Hinzufügen von Attributen und Operationen des jeweiligen Elements ermöglicht, würde dies bei BWOs wenig Sinn machen. Eine Erweiterung bzw. Verfeinerung von BWOs würde nur dann Sinn machen, wenn die in einem BWO enthaltenen Modellelemente (BWOs, BTOs) und deren Verbindungen spezialisiert und/oder ergänzt würden. Eine solche „zusammengesetzte“*

Generalisierung mit den Abhängigkeiten zwischen der Generalisierung des BWOs und den Generalisierungen der enthaltenen BTOs und BWOs ist für den Modellierer, als auch in der Realisierung zu komplex.

```
context BusinessWorkflowObject
inv:
  self.supertypes -> isEmpty and
  self.isRoot and
  self.isLeaf and
  not self.isAbstract
```

6.4.2.2 BusinessTaskObject

Ein `BusinessTaskObject` ist eine Abstraktion für alle Arbeitsaufgaben der Geschäftswelt, die einer oder mehreren Eigenschaften von `BusinessActor-Objects` (BAOs) zugewiesen sind und die von genau einem Akteur (BAO) bearbeitet werden.

BTOs sind im Business Object Metamodell nicht nur Teile von BWOs im Sinne von Arbeitsschritten – so werden sie im Allgemeinen in einem Workflow Metamodell definiert – sondern können auch als eigenständige Elemente modelliert werden. Sie modellieren generell alle Interaktionen zwischen aktiven Elementen der Geschäftswelt, den Akteuren – in der Regel Menschen – und den übrigen Business Objects. BTOs sind somit eine Zusammenführung des Konzepts der Vorgangsschritte und des Konzepts der Nutzungsfälle zu dem einheitlichen Konzept der Arbeitsaufgaben. In Kapitel 7.5 wird diese Zusammenführung auch durch die Verwendung des Konzepts Arbeitsaufgabe in verschiedenen Sichten auf ein Business Object Modell dargestellt.

Definition 6.4 (Business Task Object (BTO)) *Ein `BusinessTaskObject` ist ein `Business Object` zur `Abbildung` von Arbeitsschritten eines `Geschäftsprozesses` und von Arbeitsaufgaben der `Geschäftswelt`. Die Arbeitsschritte werden von einem Akteur bearbeitet. Deshalb sind BTOs einem `Business Actor Object` zugewiesen. Sie bilden somit alle Interaktionen zwischen aktiven Elementen der `Geschäftswelt`, den Akteuren – in der Regel Menschen – und den übrigen `Business Objects` ab.*

Ähnlich wie das BWO hat ein BTO eine Reihe vordefinierter Framework-Operationen: `resume()`, `terminate()`, `abort()`, `suspend()` und

`complete()`. Sie dienen in erster Linie zur Einbindung eines BTOs in einen BWO und werden im Abschnitt über das Package „Dynamics“ beschrieben.

Regel 19 (Namensraumregel für BusinessTaskObjects) *Ein BTO kann folgende Elemente enthalten* `DataType`*,* `PropertyAssociation`*,* `Operation`*,* `Attribute` *und* `Event`*.*

```
context BusinessTaskObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(DataType) or
    e.oclIsKindOf(PropertyAssociation) or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )
```

6.4.2.3 BusinessActorObject

Die `BusinessActorObjects` (BAOs) sind Modellelemente zur Abbildung aller aktiven Elemente der Geschäftswelt. Aktiv heißt, daß von diesen Business Objects in letzter Instanz die Initiative und die Kompetenz zur Änderung oder Abfrage des momentanen Zustands eines Business Object Systems ausgeht. Dies sind in der Regel Menschen, nämlich die Mitarbeiter der Organisation, deren Geschäftswelt auf ein Business Object Modell abgebildet wurde. Allerdings: auch Maschinen – speziell andere Softwaresysteme – können selbständig bestimmte geschäftliche Handlungen anstoßen. Sie werden ebenfalls durch BAOs modelliert.

Damit sind die BAOs im Business Object Metamodell Modellelemente zur expliziten Modellierung der Akteure, die bestimmte Arbeitsaufgaben des Systems ausführen. Das heißt im Kontext eines Business Objects Systems, das sie andere Business Objects erzeugen, löschen, ändern, miteinander verknüpfen, ausführen oder einfach nur deren Eigenschaften abfragen können. Dies ist ein eigenständiges Merkmal dieses Business Object Metamodells, denn im Gegensatz dazu werden in traditionellen Informationssystemen Akteure höchstens im Rahmen der Authentifizierung und Autorisierung getrennt vom Modell des eigentlichen Systems explizit modelliert.

Eine Ausnahme bilden dabei die Use Case Modellierung und Workflow-Systeme. Mit Use Cases werden ebenfalls Akteure – beziehungsweise deren Rollen – und

die Dienste des Systems, die diese Akteure in Anspruch nehmen, modelliert. Die Umsetzung der Use Cases in den Entwurf eines Softwaresystems sind aber bei allen bisherigen Methoden nicht systematisch. Vor allem werden nur die Nutzungsfälle mit Interaktions- oder Aktivitätsdiagrammen weiter verfeinert, die Akteure werden jedoch nicht in den Entwurf übernommen. Sie bleiben ein Artefakt der Anforderungsanalyse.

Definition 6.5 (Business Actor Object (BAO)) *Das Business Actor Object ist ein* `Business Object` *zur Abbildung aller aktiven Elemente der* `Geschäftswelt`*. Aktiv heißt, daß von diesen* `Business Objects` *in letzter Instanz die Initiative und die Kompetenz zur Änderung oder Abfrage des momentanen Zustands eines* `Business Object Systems` *ausgeht. Dies können die Mitarbeiter eines Unternehmens sein oder* `Systeme`*, die nicht Teil des* `unternehmensweiten Systems` *auf Basis von* `Business Objects` *sind.*

Die BAOs bedeuten eine neuartige Integration mehrerer Konzepte in einem Modellelement:

- **Aufgabenzuordnung und Aufbauorganisation**: Eine Zuweisung von Rollen, Bildung von organisatorischen Strukturen oder die Zuweisung von bestimmten Kompetenzen erfolgt über die Beziehung `Association` zwischen BAOs und BEOs. Das heißt, Rollen, Organisationsschemata oder sonstige Eigenschaften werden mithilfe von BEOs modelliert und den BAOs zugewiesen. Dies ermöglicht eine größtmögliche Flexibilität gegenüber verschiedensten Organisations- und Aufgabenformen.

- **Session- und Autorisierungsverwaltung**: Dadurch, daß jeder Benutzer des Systems als Business Object modelliert wird, kann eine Session- und Autorisierungsverwaltung für Anwendungen des Business Object Systems in einheitlicher Form realisiert werden. Zugriffsrechte auf andere Business Objects werden dadurch modelliert, daß – ähnlich wie in der Aufgabenzuordung – mithilfe von `Associations` die Credentials als BEOs modelliert und den jeweiligen BAOs zugewiesen werden.

- **Kapselung anderer Systeme**: Eine Hauptforderung an ein Business Object Modell ist es, die Abbildung der Geschäftswelt – das eigentliche Modell – zu trennen von technischen Konzepten, die notwendig sind, dieses Modell mithilfe von Informationstechnologie zu automatisieren. Allerdings existieren Situationen, die es notwendig machen, bestimmte Elemente in das Modell zu integrieren, die nicht Teil der Geschäftswelt sind. Sollen zum

Beispiel Altsysteme mit einem Business Object System gekoppelt werden, so kann man entweder das komplette System reengineeren, um die eigentlichen Business Objects zu erhalten, oder man kapselt das System zum Teil oder als Ganzes. Die erste Strategie ist sehr aufwendig und kaum mehr möglich, wenn außer dem Quelltext keine Information über das System vorhanden ist [Arn91]. In bestimmten Fällen – zum Beispiel wenn das Altsystem nur Geschäftsprozesse oder Aufgaben des Business Object Systems triggert – genügt es, diese Aufrufe durch die in diesem Abschnitt eingeführten BAOs zu repräsentieren.

Regel 20 (Namensraumregel für BusinessActorObjects) *Ein BWO kann folgende Elemente enthalten* `Operation`*,* `Attribute` *und* `Event`*.*

```
context BusinessActorObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )
```

6.4.3 Dynamik

Im folgenden werden die für die Integration von arbeitsteiligen Vorgängen notwendigen Modellelemente des Metamodells vorgestellt, die die Ablauforganisation von Workflows spezifizieren. Sie ergänzen das im Abschnitt 6.3.2 beschriebene Package „Dynamics“.

Die Ablauforganisation von Workflows wird nach [Jab95] bestimmt durch Kontrollflußkonstrukte, die Workflows und Aktivitäten der Workflows miteinander verbinden und bestimmen, welche kausalen Abhängigkeiten zwischen ihnen bestehen. Im Business Object Metamodell wird die Ablauforganisation ausgedrückt durch die Modellelemente `Flow` und `Action`.

6.4.3.1 Flow

Das Modellelement `Flow` definiert einen möglichen Kontrollflußübergang zwischen

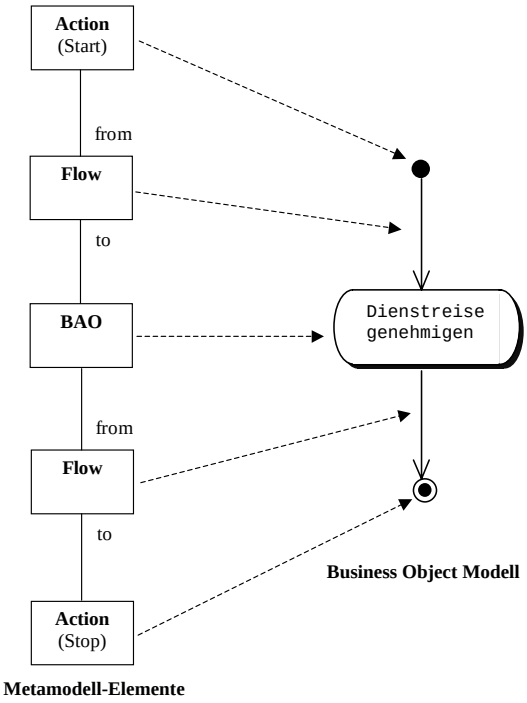

Abbildung 6.8: Metamodellelemente Flow und Action und ihre Abbildung

- zwei Business Activity Objects,
- einem Business Workflow Object und einem Business Activity Object (Aufruf eines Subworkflows) und
- einem Business Activity Object bzw. einem Business Workflow Object und einer `Action` (Verzweigung, bedingte Ausführung, Start- und Endpunkt).

Ein Kontrollflußübergang bedeutet, daß die Aktivität des BAOs bzw. des Action-Elements, von dem der `Flow` ausgeht, gerade beendet wurde und im Anschluß die Aktivität des BAOs bzw. des Action-Elements auf den der `Flow` verweist, gestartet wird.

In Abbildung 6.8 ist die Abbildung der Metamodellelemente `Flow`, `Action` und `Business Activity Object` auf ein einfachen Workflow mit einem Start- und Endpunkt sowie einer Aktivität dargestellt.

Kontrollflußkonstrukte eines Workflows sind als `Flow`-Modellelemente im Namensraum des entsprechenden `Business Workflow Objects` (siehe Abschnitt 17 enthalten.

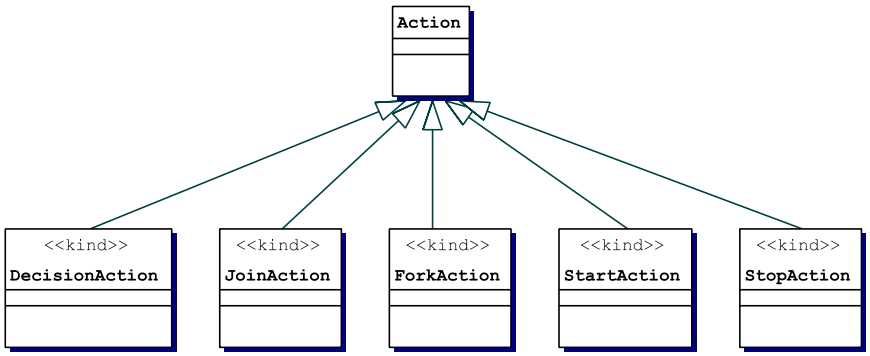

Abbildung 6.9: Die verschiedenen Actionelemente

Referenzen

fromClassifier Referenz auf die zugehörige Aktivität, nach deren Beendigung die Kontrolle auf eine folgende Aktivität `toClassifier` übergehen soll.

toClassifier Verweis auf die Aktivität, die als nächstes die Kontrolle erhalten soll, wenn die Aktivität `fromClassifier` beendet ist.

Regel 21 (Flow-Partner) *Ein `Flow` kann wie oben beschrieben nur zwischen BAOs, BWOs und `Actions` erfolgen.*

```
context Flow
inv:
  (self.fromClassifier.oclIsKindOf(BusinessActivityObject) or
  self.fromClassifier.oclIsKindOf(BusinessWorkflowObject) or
  self.fromClassifier.oclIsKindOf(Action)) and
  (self.toClassifier.oclIsKindOf(BusinessActivityObject) or
  self.toClassifier.oclIsKindOf(BusinessWorkflowObject) or
  self.toClassifier.oclIsKindOf(Action))
```

6.4.3.2 Action

Mit `Action`-Elementen werden unterschiedliche Kontrollflußkonstrukte für Workflows beschrieben. Abbildung 6.9 zeigt die Konstrukte:

- DecisionAction: Zur Modellierung von konditionalen Abläufen zwischen mehreren Aktivitäten in einem Workflow.

- ForkAction: Startpunkt für parallele Abläufe in Workflows.
- JoinAction: Zusammenführung von parallelen Abläufen.
- StartAction: Startpunkt eines Workflows. Es darf höchstens einen Startpunkt pro Workflow geben. Sollen mehrere Abläufe gleichzeitig in einem Workflow starten, so ist eine `ForkAction` nach der `StartAction` zu verwenden.
- StopAction: Endpunkt eines Workflows. Es kann mehrere `StopActions` pro Workflow geben.

Attribute

kind Mit `kind` wird festgelegt, welches der oben genannten Kontrollflußkonstrukte modelliert werden soll. Mögliche Werte sind: „start“, „stop“, „if“, „fork“ und „join“.

6.5 Zusammenfassung

In den vorangegangenen Abschnitten wurde das Business Object Metamodell als Basis für alle Elemente des in dieser Arbeit vorgestellten methodischen Frameworks zur Entwicklung offener, unternehmensweiter Systeme entwickelt. Es erfüllt die in Kapitel 3 aufgestellten Kriterien, wobei besonders auf die in anderen Ansätzen nicht vorhandene Integration von arbeitsteiligen Vorgängen erstmals gelöst wurde.

Folgende Ziele wurden mit dem Business Object Metamodell erreicht:

- Die Modellelemente des Metamodells erlauben die Abbildung von Teilen einer Geschäftswelt in ein Business Object System.
- Sowohl strukturelle Abhängigkeiten, als auch dynamische Vorgänge können im Business Object Modell ausgedrückt werden. Dabei sind keine Abhängigkeiten zu Elementen außerhalb der Geschäftswelt notwendig.
- Das Metamodell ist nicht an ein bestimmtes Geschäftsfeld oder einen bestimmten Anwendungsbereich gebunden, sondern allgemein einsetzbar.
- Das Metamodell ist in Packages strukturiert und bietet Möglichkeiten zur nachträglichen Erweiterung.

- Durch die Spezifikation des Metamodells mit Hilfe des MOF Metamodells ist ein Werkzeugunterstützung durch MOF-konforme Werkzeuge möglich.

Die Erweiterbarkeit des Metamodells ist durch die Einführung der allgemeinen Modellelemente und die Aufteilung in drei Packages erreicht worden. Die allgemeinen Modellelemente, vor allem das Namensraumkonzept, ermöglichen eine Hinzunahme von Modellelementen, ohne, daß sich die bisherige Struktur ändern müßte. Es ist lediglich eine Anpassung der (OCL-)Regeln zu den Namensräumen nötig. Falls diese in einem Werkzeug für das Business Object Modell nicht „fest verdrahtet“, sondern änderbar sind, muß auch am Werkzeug nicht viel geändert werden.

Damit das Metamodell von einer Reihe auf dem Markt verfügbarer Modellierungswerkzeuge realisiert werden kann, wurde eine weitreichende Werkzeugfähigkeit des Metamodells geachtet. Es existieren auf dem Markt mehrere kommerzieller, MOF-konformer Modellierungswerkzeuge, die zumindest XMI [OMG99c] importieren und exportieren können, wie Unisys Modeller, Rational Rose, Softera Soft Modeler, Together Control Center, Platinum Paradigm Plus und Aonix Software Through Pictures. XMI ist ein MOF-konformes Austauschformat auf Basis von XML für UML-Modelle. Es sind bereits Forschungsprototypen wie ARGO UML [RR00] mit OCL-Interpreter zur Implementierung von Konsistenzregeln verfügbar. Eine Anpassung solcher Werkzeuge an das Business Object Metamodell ist – je nach Konfigurationsframework des Tools – einfach machbar.

Und schließlich wurde das Business Object Metamodell unabhängig von allen Middleware- oder Workflow-Infrastrukturen gehalten. Dies ist vor allem dann wichtig, wenn ein Business Object Modell auf Basis des Metamodells nicht schon beim nächsten Technologiewechsel veraltet sein soll und geändert werden muß.

Nachdem nun das Business Object Metamodell definiert wurde, können jetzt alle anderen Teile des Frameworks – die wie in Abbildung 1.1 in der Einleitung dargestellt – entwickelt werden. Im nächsten Kapitel werden Beschreibungstechniken entwickelt, die auf dem Business Object Metamodell basieren. Im folgenden Kapitel wird ein Vorgehensmodell für die Entwicklung unternehmensweiter Anwendungen erarbeitet. Im Kapitel 9 werden schließlich eine Realisierung des Metamodells in einem Modellierungswerkzeug sowie eine Abbildung auf eine Infrastruktur dargestellt.

Kapitel 7

Beschreibungstechniken

Modelle sind ein wichtiger Bestandteil eines systematischen Vorgehens in der Softwareentwicklung. Dabei ist die konkrete Darstellung der Modelle entscheidend. Vor allem durch die Entwicklung und den verstärkten Einsatz intuitiver graphischer Beschreibungstechniken können Systeme in verschiedenen Phasen aus unterschiedlichen Sichtweisen in einer abstrakten und problemorientierten Weise diskutiert und entworfen werden, ohne das eine Codezeile geschrieben werden müßte. Im Gegenteil: Durch die modellbasierte Entwicklung von Systemen wird gegenüber einer Try-and-Error Programmierstrategie ein höheres Qualitätsniveau in kürzerer Zeit erreicht.

Grundsätzlich ließe sich ein Business Object System allein in der Sprache des im vorigen Kapitel vorgestellten Metamodells entwerfen. Dazu könnte eine textuelle Instanz des Metamodells genutzt werden. Allerdings wäre dieses Unterfangen unter Umständen genauso mühsam und fehleranfällig, wie das System direkt in einer Programmiersprache zu beschreiben. In diesem Kapitel wird als Teil des methodischen Frameworks ein Satz an Beschreibungstechniken für Business Object Modelle vorgestellt.

Die hauptsächlichen Ergebnisse dieses Kapitels sind:

- Ein integrierter Satz an Techniken, die die Möglichkeit bieten, verschiedene Sichten auf ein Business Object System zu erhalten.
- Abdeckung aller Systemkonzepte des Business Object Metamodells durch die mit Beschreibungstechniken spezifizierbaren Modelle.
- Alle Diagrammarten wurden auf das gemeinsame Business Object Meta-

modell abgebildet und die Bedeutung der Modellelemente auf struktureller Ebene festgelegt.

- Im Gegensatz zu anderen Ansätzen als Profil der UML definiert und damit mithilfe UML-kompatible Werkzeuge modellierbar.
- Eigenständige, integrierte Modellierungsmöglichkeit arbeitsteiliger Vorgänge in UML.

7.1 Übersicht

In Abschnitt 5.2.2 wurden bereits verschiedene Sichten vorgestellt, die bei der traditionellen Entwicklung von Anwendungssystemen eine Rolle spielen:

- Logische Sicht
- Implementierungssicht
- Prozeßsicht
- Laufzeitsicht
- Nutzungsfallsicht

In einem Business Object Modell sind aber aufgrund der strikten Trennung von Abbildung der Geschäftswelt und Infrastruktur- oder Laufzeitdetails nur die logische Sicht und mit Einschränkungen die Nutzungsfallsicht von Bedeutung.

Implementierungssicht, Prozeßsicht und Laufzeitsicht sind Sichten, die im Business Object Metamodell keine Rolle spielen. Logische Sicht und Nutzungsfallsicht werden im Metamodell beschränkt auf eine Teilmenge der Business Architektur, das heißt die Elemente der Geschäftswelt und deren Beziehungen untereinander.

In Abschnitt 5.2.2 wurden bereits die Sichten, die mit dem Metamodell ausdrückbar seinen sollen, definiert. In diesem Kapitel werden nun geeignete Beschreibungstechniken für die Modelle der einzelnen Sichten entwickelt. In Abbildung 7.1 ist das 4+1 Sichtenmodell aus Abschnitt 5.2.2 erweitert mit den jeweils gewählten Techniken. In der

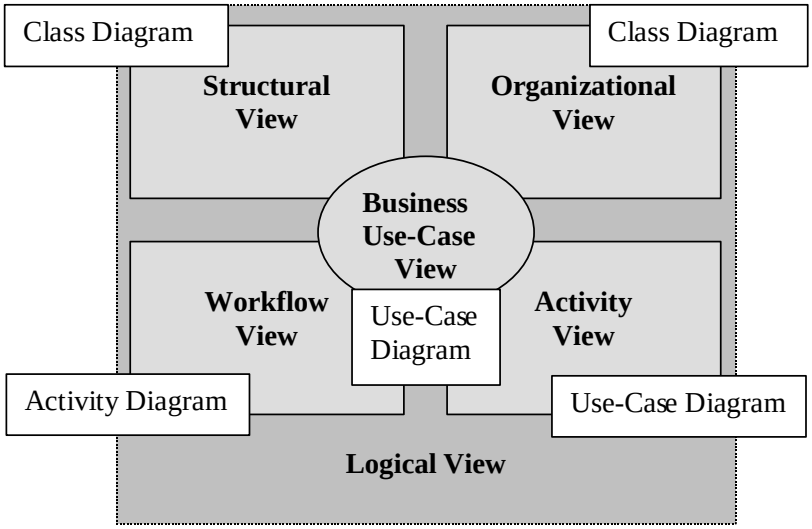

Abbildung 7.1: Das 4+1 Sichtenmodell und die jeweiligen Beschreibungstechniken

- *Geschäftsvorfallsicht* (Business Use Case View), werden die Geschäftsvorfälle eines Geschäftsbereiches ebenfalls mit Use Case Diagrammen dargestellt,
- *Struktursicht* (Structural View), wird die statische Struktur der Business Objects beschrieben mit Klassendiagrammen,
- *Aufgabensicht* (Activity View), werden die nicht arbeitsteiligen Aufgaben und die diese ausführenden Akteure mit Use Case Diagrammen beschrieben,
- *Vorgangssicht* (Workflow View), werden die dynamischen und hierarchischen Abhängigkeiten von arbeitsteiligen Vorgängen mit Aktivitätsdiagrammen beschrieben,
- *Organisationssicht* (Organizational View), in der die Akteure eines Unternehmens und ihre Einordnung in die funktionalen wie organisatorischen Einheiten der Unternehmung, dazu die möglichen Rollen, die die Akteure einnehmen können, werden ebenfalls Klassendiagramme verwendet.

Damit die Modelle der oben genannten Sichten einheitlich und unzweideutig ein gültiges System auf Basis des Business Object Metamodells ergeben, müssen die Diagrammelemente eine genaue Bedeutung im Metamodell haben. Dazu werden die jeweiligen Metamodellelemente des Diagrammtyps auf das Business

Object Metamodell abgebildet und falls nötig auf eine bestimmte Teilmenge eingeschränkt. Dies geschieht in dieser Arbeit durch die Angabe eines sogenannten Profils der UML. Im nächsten Abschnitt wird die UML kurz vorgestellt und gezeigt, welche Erweiterungsmöglichkeiten genutzt werden können, um eine Teilmenge der UML-Diagrammtypen zu einer Modellierungstechnik für Business Object Modelle anzupassen und auf das Business Object Metamodell abzubilden.

7.2 Unified Modeling Language

Die UML ist eine graphisch-orientierte Modellierungssprache mit dem Ziel, die Analyse, den Entwurf und die Entwicklung von Softwaresystemen zu unterstützen. Sie beinhaltet einen Satz von verschiedenen Beschreibungstechniken, die es erlauben, Eigenschaften eines Systems aus verschiedenen Sichten zu spezifizieren. Ausgangspunkt der Entwicklung der UML war 1996 die Ausschreibung der OMG [OMG96] für einen Standard einer generischen, einheitlichen Sprache zur Entwicklung von Softwaresystemen.

Zu dieser Zeit existierten viele Modellierungssprachen mit teils unterschiedlichen, teils gemeinsamen Konzepten, die untereinander – vor allem im Hinblick auf die verwendeten Modellierungswerkzeuge – wenig kompatibel waren. Die UML war als Zusammenführung der damals am weitest verbreiteten Modellierungssprachen Booch [Boo91], OMT [RBP+91] und OOSE [JCJO92] der am meisten akzeptierte Vorschlag auf die Ausschreibung und wurde 1997 von der OMG als Standard (UML 1.1) verabschiedet [OMG99a].

Verglichen mit den Möglichkeiten der Vorgänger der UML (Booch, OMT, OOSE), wurden mit der UML weitere, neue Elemente eingeführt: Erweiterungsmechanismen (werden im Abschnitt 7.2.2.2 vorgestellt), Unterstützung der Modellierung von Patterns und Collaborations, Aktivitätsdiagramme, Interfaces und Komponenten, sowie die Object Constraint Language (OCL).

Die UML ist seit ihrer Standardisierung – mit Ausnahme von Bereichen wie die Modellierung von Echtzeit- , eingebetteten oder Workflowmanagement-Systemen – eine Standard-Modellierungssprache für die Industrie, nicht zuletzt auch deshalb, weil es inzwischen eine konkurrenzlose Anzahl von kommerziellen Werkzeugen zur Modellierung von UML gibt.

Trotz dieser Akzeptanz gibt es allerdings noch einige kritische Punkte, die Thema zahlreicher Konferenzen [FR99, SK98, KRS98, KR98] zum Thema UML

sind. Hauptkritikpunkt ist die mangelnde Integration der einzelnen Beschreibungstechniken. Zwar wird die gesamte UML durch ein Metamodell auf Basis der MOF beschrieben, doch damit wird im wesentlichen nur die abstrakte Syntax der „Sprache“ UML beschrieben. Die Bedeutung der Konzepte und wie diese Konzepte aufeinander abgebildet werden können bleibt oftmals unklar [Rum98]. Das hat auch Auswirkungen auf die UML-Werkzeuge: Die Diagrammarten können alle dargestellt und bearbeitet werden, Navigationsmöglichkeiten zwischen diesen oder eine modellweite Konsistenzprüfung fehlen jedoch.

Auch die einzelnen UML-Diagrammarten sind nicht unumstritten. Im folgenden Abschnitt werden speziell die für die Business Object Modellierung ausgewählten UML-Diagrammtypen vorgestellt und ihre kritischen Punkte diskutiert. Für eine ausführliche Darstellung der UML wird auf [OMG99a, BRJ99, RJB98] verwiesen.

7.2.1 Diagrammtypen

Wie oben schon erwähnt, ist die UML eine Zusammenführung von drei verschiedenen, bereits etablierten Modellierungstechniken. Da zwischen diesen nur wenig Überschneidung bezüglich der verwendeten Diagrammarten bestand, ist die Auswahl in der UML an unterschiedlichen Beschreibungstechniken und damit das Vokabular zur Beschreibung von Softwaresystemen entsprechend groß. Abbildung 7.2 zeigt die verschiedenen Konzepte – eingeteilt in die drei Kategorien Struktur, Verhalten und Kommentar – und die neun Diagrammarten der UML.

Für das Profil der UML für die Modellierung von Business Object Systemen wurden die Use-Case-, Klassen- und Aktivitätsdiagramme der UML ausgewählt.

7.2.1.1 Use-Case-Diagramm

In der UML-Spezifikation [OMG99a] wird ein Use-Case-Diagramme wie folgt definiert:

> Use Case diagrams show actors and use cases together with their relationships. The use cases represent functionality of a system or a classifier, like a subsystem or a class, as manifested to external interactors with the system or the classifier.

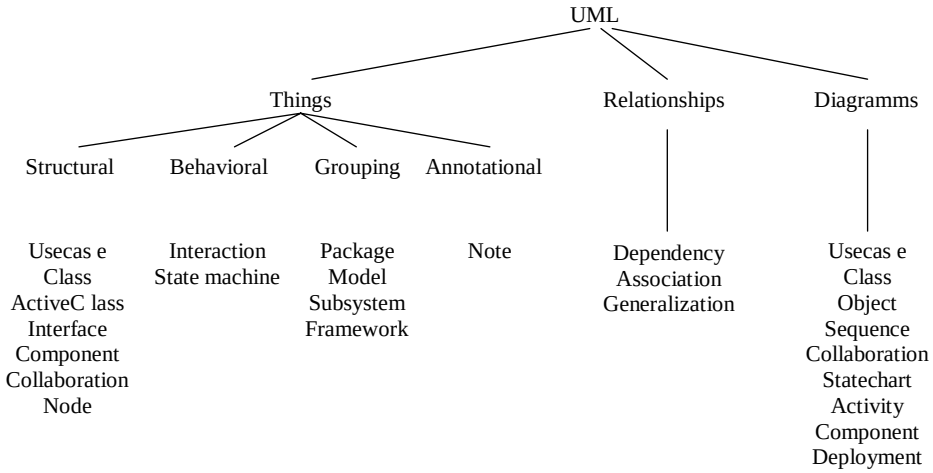

Abbildung 7.2: Vokabular und Diagrammarten der UML (aus [JBR98])

Abbildung 7.3 zeigt die möglichen Elemente eines Use-Case-Diagramms: Use Cases stehen für die Anwendungsmöglichkeiten eines Systems in einer Black-Box-Sicht. Sie sind Stellvertreter für funktional in sich geschlossene Kommunikationsabläufe zwischen einem Benutzer, der eine anwendungsabhängige Rolle (Actor) einnimmt, und dem System (SystemBoundary). Funktional abgeschlossen heißt, daß der Benutzer nach Ablauf der Interaktion im Rahmen des Use Cases eine in seinem Anwendungskontext eigenständige Aufgabe abgeschlossen bzw. bearbeitet hat. Dies wird in der Praxis oft übersehen und es werden zu feingranulare Use Cases modelliert, in denen bereits die einzelnen Schritte einer Aufgabe modellieren sind und nicht die Aufgabe selbst.

Ein Actor (Benutzer) nimmt die Dienste eines Systems in einer anwendungsabhängigen Rolle in Anspruch. Eine solche Rolle könnte zum Beispiel „Sachbearbeiter“ oder „Administrator“ sein. Ein konkreter Benutzer kann dabei mehrere Rollen einnehmen. Außerdem kann ein Benutzer eines System auch ein anderes System sein.

Durch SystemBoundary wird eine Systemgrenze festgelegt. Dieses Element ist optional und wird dann verwendet, wenn man die Use Cases eines großen Systems verschiedenen Subsystemen zuordnen möchte.

Daneben gibt es noch einige Elemente zur Verknüpfung von Use Cases und Actors: Die Verbindungen Communicates zwischen Use Cases und Actors, wie zum Beispiel zwischen „Actor1“ und „UseCase1“, ist die Voraussetzung, daß eine bestimmte Rolle einen Use Case überhaupt in Anspruch nehmen darf. Die Ei-

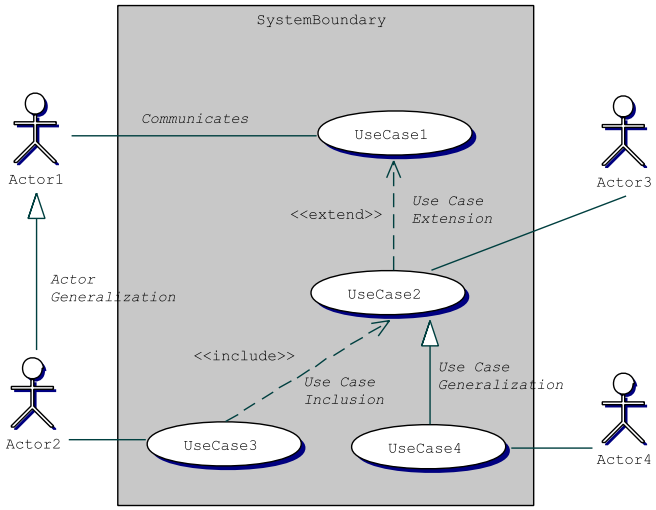

Abbildung 7.3: Abstraktes Use-Case-Diagramm

genschaften einer Rolle, das heißt die möglichen Communicates-Beziehungen, können auf eine andere Rolle vererbt werden (Actor Generalization). Die Benutzer, die diese Rolle einnehmen, können damit alle Use Cases der generalisierten Rolle in Anspruch nehmen. Zwei Use Cases können in einer Generalisierungs-Relation (Use Case Generalization) zueinander stehen. Laut UML-Spezifikation ist damit ein Use Case eine spezielle Form des generalisierten Use Cases. Was das in der Praxis jedoch bedeutet, bleibt unklar.

Und schließlich können Use Cases untereinander in einer Abhängigkeitsrelation stehen: Eine Use Case Inclusion bedeutet, daß einer der beiden Use Cases, daß heißt, die Aufgabe, die ein Benutzer damit bearbeiten kann, in einen anderem Use Case vollständig enthalten ist. Auch hier ist nicht klar, was dies in der Praxis bedeutet. Es wiederspricht der eigentlichen Definition von Use Cases als funktional in sich geschlossene Kommunikationsabläufe. Ein erweiterter Use Case (Use Case Extension), stellt eine Variante eines anderen Use Cases dar. Die Variante, das heißt ein geänderter Kommunikationsablauf, tritt unter bestimmten Bedingungen in Kraft, die durch die sogenannten Extension Points beschrieben werden.

Use-Cases-Diagramme werden meist am Anfang einer Softwareentwicklung eingesetzt, um die grundsätzlichen Akteure und Aufgaben eines Systems darzustellen. Die weitere, systematische Umsetzung von Use Cases in den Entwurf eines Systems ist derzeit aber nicht Stand der Praxis. Es mangelt vor allem an einem methodischen Vorschlag, wie die im Diagramm dargestellten Elemente auf die Elemente anderer Modellierungstechniken abgebildet werden sollen.

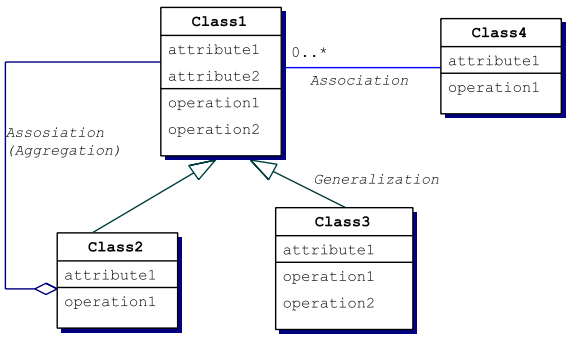

Abbildung 7.4: Abstraktes Klassendiagramm

7.2.1.2 Klassendiagramm

Das Klassendiagramm steht bei der UML – wie in den meisten objektorientierten Methoden – im Mittelpunkt der Modellierung. Es wird in der Analyse, im Design und werkzeugunterstützt auch bei der Generierung des Quelltextes eines Systems verwendet.

Ein UML-Klassendiagramm, wie in Abbildung 7.4 dargestellt, beschreibt die statische Struktur eines System bestehend aus Klassen, deren Attribute und Operationen, sowie Generalisierungsrelationen und Assoziationen zwischen diesen. Klassen stehen für weiter nicht teilbare Entitäten der Anwendungs- und Systemwelt. Durch ihre Attribute wird ein diskreter Zustandsraum für die Entitäten definiert. Die Operationen ermöglichen eine Manipulation dieses Zustandsraumes. Assoziationen verbinden zwei Klassen und ermöglichen eine gegenseitige Manipulation und Navigation.

Die Generalisierungsrelation ermöglicht eine Hierarchisierung eines Klassensystems, indem Attribute, Operationen und Assoziationen der generalisierten Klasse von der von dieser abgeleiteten Klasse übernommen werden können.

Klassendiagramme sind schon eine seit den Anfängen der objektorientierten Analyse bekannte und seitdem immer weiter konsolidierte Modellierungstechnik. Die UML-Klassendiagramme übernehmen die bekannten Elemente der bereits existierenden Methoden, die sich bis auf Darstellungsfragen in ihrer Ausdrucksmächtigkeit weitgehend ähnelten. So spielt das UML-Klassendiagramm bei den Beiträgen mit Kritik an der UML auch eine untergeordnete Rolle. Eine Ausnahme bildet die bis heute nicht eindeutig geklärte Bedeutung der verschiedenen Assoziationsarten, wie Aggregation oder Komposition [SLPFE98].

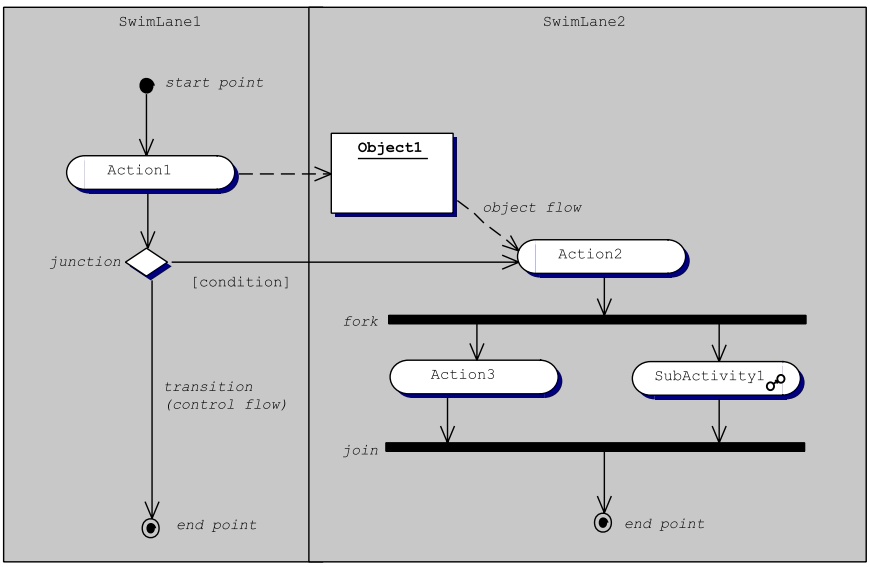

Abbildung 7.5: Abstraktes Aktivitätsdiagramm

7.2.1.3 Aktivitätsdiagramm

In der UML ist das Aktivitätsdiagramm eine von allen anderen Diagrammen weitgehend unabhängige Modellierungstechnik zur Beschreibung von prozeduralen Abläufen. Die Wurzeln dieses Diagrammtyps und seiner Konzepte liegen auch nicht in den objektorientierten Methoden, sondern in den Daten- und Kontrollflußdiagrammen der strukturierten Analyse [AG90] und den Ereignisdiagrammen für die Prozeßmodellierung [KNS92].

Die grundlegenden Elemente eines Aktivitätsdiagramms sind Aktionen, Unteraktivitäten (Stellvertreter für andere Aktivitätsdiagramme) und Transitionen. In Abbildung 7.5 werden drei Aktionen und eine Unteraktivität mit insgesamt zehn Transitionen dargestellt. Jede Transition bedeutet einen Kontrollflußübergang zwischen zwei Aktionen/Unteraktivität oder einer Aktion/Unteraktivität und einem speziellen Kontrollflußkonstrukt. Solche speziellen Konstrukte können sein: Start- oder Endpunkte des Kontrollflusses, Entscheidungsknoten oder Elemente zum Starten oder Zusammenführen von parallelen Abläufen.

Durch die Angabe von Verantwortlichkeitsbereichen, den sogenannten „Swim Lanes“, können die Aktionen/Unteraktivitäten auf ein oder mehrere Akteure verteilt werden. In der UML-Spezifikation wird nicht deutlich, inwieweit die Verantwortlichkeitsbereiche in andere Modellelemente (aktive Klassen o.ä.) abgebildet werden können oder mit ihnen in Beziehung stehen. Deshalb bleibt dieses Ele-

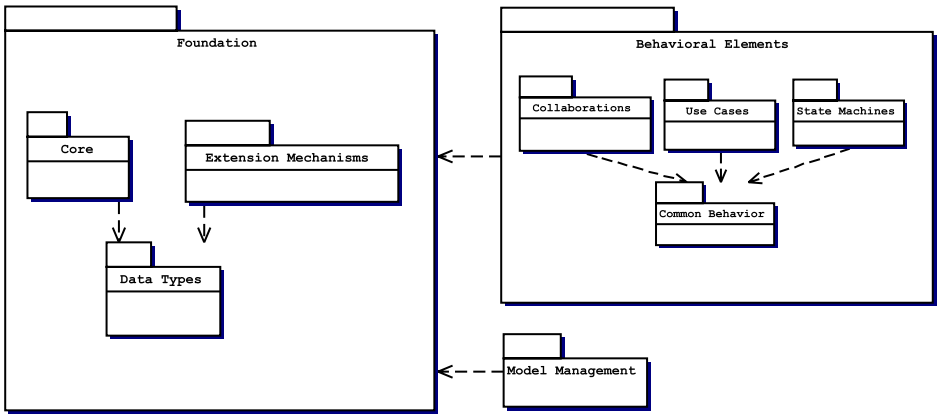

Abbildung 7.6: Package-Struktur des UML-Metamodells

ment nur ein graphisches Hilfsmittel.

Eine Besonderheit der UML-Aktivitätsdiagramme ist die Einführung von Objekten im Rahmen von Datenfluß-Abhängigkeiten. In der Abbildung 7.5 dargestellt durch „Object1“ und dem Datenfluß von „Action1“ zu „Action2“. Allerdings äußert sich die UML-Spezifikation nicht eindeutig zu der Semantik dieser Objektfluß-Konstrukte. Es wird zum Beispiel nicht klar, welcher Pfad Vorrang hat, wenn zwischen zwei Aktionen sowohl ein Datenfluß als auch ein Kontrollfluß besteht.

7.2.2 UML-Metamodell

Die UML wird semiformal definiert durch die Angabe eines Metamodells [OMG99a], OCL-Ausdrücken und textuellen Beschreibungen. Semiformal deshalb, da die abstrakte Syntax der Modellierungstechniken durch MOF-Modelle und OCL-Ausdrücke formal, die Bedeutung der jeweiligen Konzepte jedoch in Englisch, also nicht-formal beschrieben wird.

7.2.2.1 Struktur

Das Metamodell ist unterteilt in drei Packages: Foundation, Behavioral Elements und Model Management. Abbildung 7.6 zeigt die Package-Struktur und die einzelnen Unter-Packages. Die Konzepte der UML, die bereits in Abbildung 7.2 größtenteils dargestellt wurden, sind auf diese Packages aufgeteilt:

- Core: Class, Attribute, Operation, Association, Relationsship, Component, Node, Comment
- Data Types: Verschiedene vordefinierte Datentypen
- Extension Mechanisms: Constraint, Stereotype, Tagged Value
- Common Behavior: Signal, Operation, Action
- Collaborations: Collaboration, Interaction, Message
- Use Cases: Actor, Use Case
- State Machines: State, Event, Signal, Transition, Activity, Object Flow
- Model Management: Package, Model, Subsystem

Für eine umfassende Darstellung des Metamodells wird auf [RJB98] verwiesen. Für diese Arbeit wichtig sind die Erweiterungsmechanismen des Packages Extension Mechanism, die im folgenden dargestellt werden.

7.2.2.2 Erweiterungsmechanismen

Die Möglichkeit, die UML durch spracheigene Elemente an spezielle Anwendungsgebiete oder Einsatzzwecke anpassen zu können, ist ein Hauptgrund für die deren Auswahl als Basis der Business Object Modellierungssprache. Da die Erweiterungselemente von vornherein im Standard enthalten sind, wird ermöglicht, bestehende UML-Werkzeuge in in einer standardisierten Weise zu erweitern.

Zur Anpassung der UML stehen drei Modellelemente zur Verfügung: Stereotypen, Tags und Constraints.

Stereotypen Der wichtigste Erweiterungsmechanismus ist das Konzept der Stereotypen. Sie bieten eine Möglichkeit, während der Modellierung Unterklassen von allen vorhandenen UML-Metamodellklassen mit neuen Attributen und zusätzlicher Semantik zu definieren. Ein Stereotyp kann optional durch eine neue graphische Repräsentation dargestellt werden. Im Normallfall erscheint der Name des Stereotyps in französischen Anführungszeichen (guillemets, « »). Genauer spezifiziert wird ein Stereotyp im Regelfall durch erforderliche Tagged Values, die weiter unten beschreiben werden. Hauptaufgabe der Stereotypen ist es, den Werkzeugen die mit einem UML-Modell

arbeiten – zum Beispiel Code-Generatoren – Hinweise auf eine besondere Verarbeitung der mit Stereotypen gekennzeichneten Elemente zu geben.

Tagged Values Tag Definitionen spezifizieren neue Eigenschaften, die zu einem Modellelement hinzugefügt werden können. Die tatsächlichen Werte dieser Eigenschaften bei den konkreten Modellelementen wird durch Tagged Values (Eigenschaft-Wert-Paare) ausgedrückt. Dies können Werte einfacher Datentypen oder Referenzen auf andere Modellelemente sein. Tag Definitions sind Meta-Attribute (Ebene M2), Tagged Values entsprechen Werten die den Elementen des Modells (Ebene M1) zugewiesen werden. Tagged Values können zum Beispiel genutzt werden, um einem Code-Generator zusätzliche Informationen (Optimierungsstufe o.ä.) mitzuteilen. Sie werden als Eigenschaft-Wert-Paare in geschweiften Klammern (z.B. status=released) den jeweiligen Modellelementen hinzugefügt.

Constraints Durch ein Constraint können zusätzliche Konsistenzbedingungen über das Modell ausgedrückt werden. Über die Art der logischen Ausdrücke macht die UML-Spezifikation keine Aussage. Es können textuelle Ausdrücke in natürlicher Sprache sein, oder formale Ausdrücke, zum Beispiel in OCL. Letzteres ist natürlich dann sinnvoll, wenn die Konsistenzbedingungen werkzeugunterstützt ausgewertet werden sollen. Ein Constraint wird im Modell durch einen Ausdruck in geschweiften Klammern (z.B. complete) hinzugefügt.

Einen Satz der oben dargestellten Erweiterungselemente für ein Modell eines speziellen Anwendungsbereiches bezeichnet man als Profil. Darüberhinaus kann ein Profil auch die im Modell enthaltenen Elemente durch Angabe eines Subsets des kompletten UML-Metamodells einschränken.

In den folgenden Abschnitten wird das UML-Profil zur Modellierung von Business Object Systemen vorgestellt.

7.3 Geschäftsvorfallsicht

In der Geschäftsvorfallsicht werden die Geschäftsvorfälle eines Geschäftsbereiches betrachtet. Im Gegensatz zu den Business Use Cases von Jacobson [JGJ97], die ein komplettes Geschäftsmodell eines Bereiches wiedergeben, beschränken sich die Geschäftsvorfälle des Business Object Modells allein auf die Vorfälle, die auch vom Business Object System unterstützt werden.

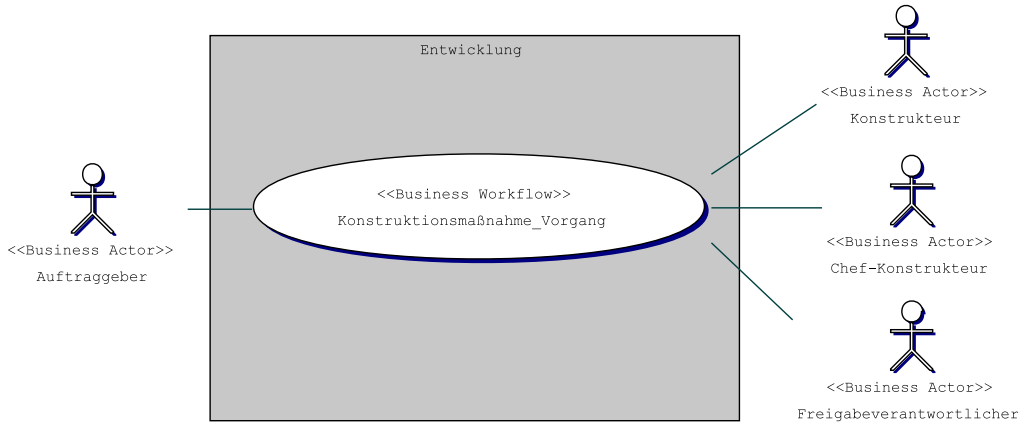

Abbildung 7.7: Beispiel eines Business-Use-Case-Diagramms

Die Geschäftsvorfallsicht ist damit an der Schnittstelle von Geschäftsprozeßmodellierung und Systementwicklung. Für die Geschäftsvorfallsicht dieser Arbeit wird vorrausgesetzt, daß bereits ein Business Process Reengineering des jeweiligen Bereiches stattgefunden hat. Aufgabe der Geschäftsvorfallsicht ist es nun, diejenige Teilmenge des Geschäftsmodells ausgehend vom Modell des Business Process Reengineering zu erhalten, die auch tatsächlich IT-unterstützt wird. Damit stellt die Geschäftsvorfallsicht eine Verbindung zwischen Geschäftsprozessen und Business Objects dar.

Zur Darstellung der Geschäftsvorfallsicht werden angepaßte Use-Case-Diagramme – die Business-Use-Case-Diagramme – verwendet mit besonderen Stereotypen für die aktiven Elemente eines Geschäftsvorfalls (Business Actors) und den Geschäftsvorfällen selbst (Business Workflows).

7.3.1 Beispiel

Das in Abbildung 7.7 dargestellte Beispiel zeigt ein Business-Use-Case-Diagramm, das Geschäftsvorfälle aus dem Bereich Konstruktion und Entwicklung eines Industriebetriebs darstellt. Das Diagramm besteht im wesentlichen aus drei Elementen: Den Business Actors, den Business Workflows und den Beziehungen zwischen diesen. Im konkreten Fall wird ein Business Workflow „Konstruktionsmaßnahme_Vorgang“ und die vier beteiligten Business Actors „Auftraggeber“, „Konstrukteur“, „Chef-Konstrukteur“ und „Freigabeberechtigter“.

Der Geschäftsvorfall „Konstruktionsmaßnahme_Vorgang“ wird dann angestoßen,

Stereotyp	angewandt auf	Bedeutung
Business Actor	UseCases::Actor	Beteiligter am Geschäftsvorfall
Business Workflow	UseCases::UseCase	IT-unterstützter Geschäftsvorfall

Tabelle 7.1: Stereotypen für Business-Use-Case-Diagramme

wenn ein Produkt geändert werden muß, zum Beispiel aufgrund einer Gesetzesänderung. Der Auftraggeber beauftragt die Konstruktionsmaßnahme, die dann vom zuständigen Konstrukteur aufgesetzt und geplant wird. Der jeweilige Änderungsplan muß von einem Chef-Konstrukteur gegengezeichnet werden. Ist die Konstruktionsmaßnahme abgeschlossen, muß ein Freigabeverantwortlicher das geänderte Produkt freigeben, damit es in geänderter Form in Produktion gehen kann.

Das Beispiel zeigt, daß es in der Geschäftsvorfallsicht nur darauf ankommt, die IT-unterstützten Geschäftsvorfälle und die beteiligten Akteure zu benennen, nicht aber den kausalen oder zeitlichen Ablauf des Vorgangs oder die konkretem Aufgaben der Akteure. Diese werden in der Vorgangs- und Organisationssicht verfeinert. In den Abschnitten zur Aufgabensicht und zur Vorgangsicht wird dieses Beispiel wieder aufgegriffen und die interne Sicht des Geschäftsvorfalls „Konstruktionsmaßnahme_Vorgang“ auf verschiedenen Abstraktionsebenen spezifiziert.

7.3.2 UML-Erweiterungen

Basis der Business-Use-Case-Diagramme sind UML-Use-Case-Diagramme. Dabei wurde auf die Include-, Extend- und Generalisierungsbeziehungen verzichtet, da sie – wie oben bereits erwähnt – von der Bedeutung schon für Use-Case-Diagramme problematisch sind. Das heißt für das Profil zur Business Object Modellierung: Das Package „Use Cases“ wird für die Business-Use-Case-Diagramme übernommen, mit Ausnahme der Elemente „ExtensionPoint“, „Extend“ und „Include“.

Tabelle 7.1 zeigt die beiden in das Profil für die Business Object Modellierung aufgenommenen Stereotypen.

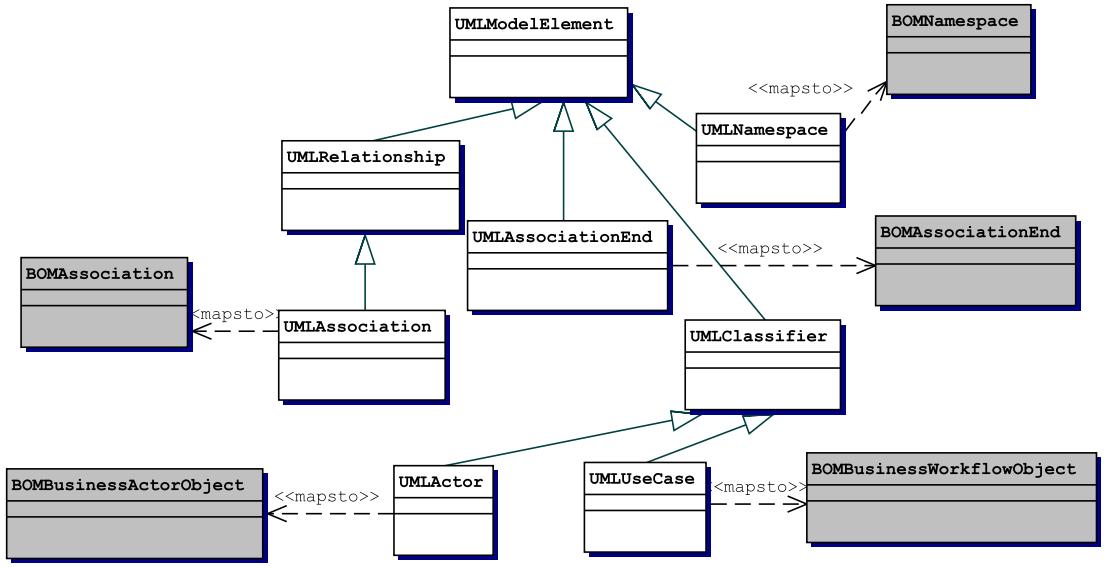

Abbildung 7.8: Mapping des Business-Use-Case-Diagramms auf das Metamodell

7.3.3 Abbildung auf das Metamodell

Die Abbildung auf das Metamodell – wie in Abbildung 7.8 dargestellt – ist aufgrund der wenigen Elemente eines Business-Use-Case-Diagramms nicht umfangreich:

- UseCases::UseCase$\oplus$«Business Workflow» $\mapsto$ Structure::BusinessWorkflowObject
- UseCases::Actor$\oplus$«Business Actor» $\mapsto$ Structure::BusinessActorObject
- Core::Association $\mapsto$ Structure::Association
- Core::AssociationEnd $\mapsto$ Structure::AssociationEnd
- Core::Namespace $\mapsto$ Base::Namespace

In der Abbildung sind zu Übersicht die Metamodellklassen des Business Object Metamodell grau hinterlegt. Zur weiteren Unterscheidung wurde vor den jeweiligen Metamodellnahmen „UML“ für die Metamodellklassen der UML und „BO“ für die Metamodellklassen des Business Object Metamodells gesetzt.

7.4 Struktursicht

Die Struktursicht steht im Mittelpunkt eines Business Object Modells. Analog zu der Modellierung objektorientierter Systeme, in denen Klassendiagramme zur Klassifizierung und Strukturierung des objektorientierten Systems eine wesentliche Rolle spielen, werden in der Struktursicht die Elemente der Geschäftswelt und ihre Beziehungen untereinander betrachtet.

Zur Modellierung der Unternehmensinformationen werden in der Struktursicht die Business Objects, also Business Entity Objects, Business Service Objects, Business Actor Objects, Business Workflow Objects und Business Activity Objects spezifiziert. Daneben werden Domänen, Proxies, Assoziationen, Attribute und Operationen sowie Events als Struktur- und Eigenschaftselemente mit den Business Objects verknüpft.

Damit deckt die Struktursicht einen Großteil der Modellelemente ab, die durch das Business Object Metamodell vorgegeben werden. Im Unterschied zum klassischen, sogenannten logischen Sicht auf ein System, wird in der Struktursicht nicht nur ein Datenmodell beziehungsweise Klassenmodell betrachtet, sondern ein umfassendes Informationsmodell als Abbild der Geschäftswelt. Wichtig ist hierbei nocheinmal zu wiederholen, daß die Struktursicht sich im Sinne des Convergent Engineering nur auf die Struktur der Abbildung der Geschäftswelt konzentriert und nicht anwendungs- oder infrastrukturspezifische Elemente enthält, wie es bei Klassendiagrammen objektorientierter Systeme spätestens beim Übergang vom Analyse- zum Designmodell der Fall ist.

Darstellt wird die Struktursicht durch Business-Object-Diagramme, eine Erweiterung der UML-Klassendiagramme. Zu den bestehenden Elementen der UML-Klassendiagramme werden zusätzlich Stereotypen zur Kennzeichnung der verschiedenen Business Object Typen verwendet und Einschränkungen bezüglich der Anwendung von Vererbungsbeziehungen und Relationen gemacht.

7.4.1 Beispiel

Das in Abbildung 7.9 gezeigte Business-Object-Diagramm stellt die Struktursicht des bereits in Abschnitt 7.3.1 eingeführten Beispiels einer Konstruktionsmaßnahme im Bereich Entwicklung und Konstruktion dar. Hier sind die Business Objects dargestellt, die bei der Durchführung einer Konstruktionsmaßnahme eine Rolle spielen. Aus Übersichtsgründen sind jedoch nicht alle in Frage kommenden Busi-

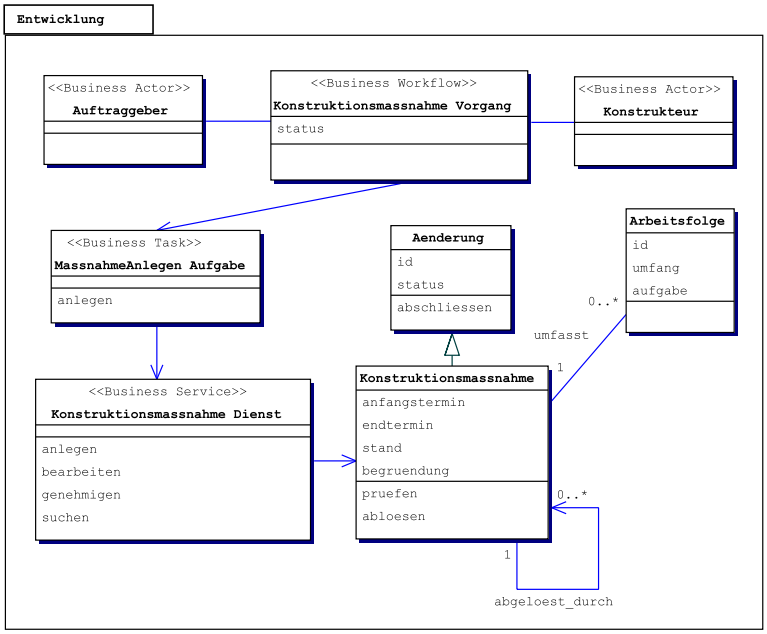

Abbildung 7.9: Beispiel eines Business-Object-Diagramms

ness Objects abgebildet.

Das Diagramm besteht aus den bekannten Elementen eines Klassendiagramms: Den Rechtecken für die klassifizierten Elemente mit deren Attributen und Operationen, sowie den Linien zwischen diesen zur Darstellung zweier Relationstypen. Vererbungsrelationen werden dardurch gekennzeichnet, daß an einem Linienende ein ausgefüllter Pfeil dargestellt ist. Dieses Linienende kennzeichnet das generalisierte Business Object. Die anderen Linien stellen Assoziationen zwischen den Business Objects dar. Eingeschlossen werden diese Elemente durch ein Rechteck, daß die Domäne der gezeigten Elemente bestimmt.

Im Beispiel sind Business Objects enthalten, die bereits in der Geschäftsvorfallsicht im Business-Use-Case-Diagramm enthalten sind: Der Geschäftsvorfall „Konstruktionsmassnahme_Vorgang“, sowie die beiden Business Actors „Auftraggeber“ und „Konstrukteur“. Im Business-Object-Diagramm wird nun deutlich, aus welchen Aufgaben der Vorgang besteht, hier exemplarisch das Business Task Object „MassnahmeAnlegen_Aufgabe“.

Hinzu kommen die eigentlichen Objekte des Geschäftsszenarios „Konstruktionsmaßnahme in der Entwicklung, die Business Entity Objects „Aenderung“, „Konstruktionsmassnahme“ und „Arbeitsfolge“. Dargestellt werden neben der Generalisierungsbeziehung zwischen „Aenderung“ und „Konstruktionsmassnah-

me“ die Assoziationen zwischen „Konstruktionsmaßnahme“ und deren jeweiligen Arbeitsabfolgen, sowie eine Assoziation „abgeloest_durch“ die modelliert, welche ursprünglichen Konstruktionsmaßnahmen im Rahmen einer Änderung von neuen Konstruktionsmaßnahmen abgelöst wurden.

Und schließlich ist im Diagramm noch das Business Service Object „Konstruktionsmassnahme_Dienst“ dargestellt, das die möglichen Geschäftstransaktionen auf den Business Entity Objects „Konstruktionsmaßnahme“ und „Arbeitsfolge“ bündelt.

Nicht im Beispieldiagramm gezeigt sind Proxies und die Erweiterungen zur Darstellung der Eventmechanismen des Business Object Metamodell. Proxies, also Stellvertreter für Business Objects, die in anderen Domänen definiert wurden, werden wie das eigentliche Business Object dargestellt mit dem zusätzlichen Stereotyp „«Proxy».

Ein Event wird als Klasse mit dem Stereotyp „«Event»“ dargestellt. Statische Eventkanäle zwischen Konsumenten (Operation) und Produzenten (Operation, Attribut) werden mit Tags hinter den jeweiligen Attribut- beziehungsweise Operationsnamen dargestellt. Mögliche Tags sind „ProducerOf“ und „ConsumerOf“. Der Wertebereich dieser Tags sind die Namen der möglichen Events des Business Object Modells. Zum Beispiel würden die Tags „ConsumerOf=StatusEvent“ hinter der Operation „genehmigen“ des Business Service Objects „Konstruktionsmassnahme_Dienst“ und „ProducerOf=StatusEvent“ hinter dem Attribut „status“ des Business Entity Objects „Aenderung“ einen Eventkanal zwischen diesen Business Objects etablieren. Falls sich der Wert des Attributs „status“ ändert, wird die Operation „genehmigen“ mit einer Instanz des Events „StatusEvent“ aufgerufen. Zuvor muß sich jedoch zur Laufzeit des Systems eine Instanz des Business Service Objects mithilfe der Framework-Operation „register“ bei einer oder mehreren Instanzen des Business Entity Objects „Aenderung“ für den Event registrieren lassen.

7.4.2 UML-Erweiterungen

Wie oben schon erwähnt, sind UML-Klassendiagramme die Basis der Business-Object-Diagramme. Ihre Modellierungskonzepte sind ausreichend, um die Struktursicht darzustellen. Elemente, die für die Struktursicht nicht relevant sind, werden nicht in das Profil für das Business Object Modell übernommen. Nicht relevant sind: Interfaces (Core::Interface) und Abhängigkeits-Relationen (Core::Dependency, Core::Binding, Core::Usage und Core::Permission). Auch das

Stereotyp	angewandt auf	Bedeutung
Business Service	Core::Class	Geschäftstransaktionen
Business Task	Core::Class	Arbeitsaufgabe
Business Actor	Core::Class	Beteiligter am Geschäftsvorfall, Zuständiger für eine Arbeitsaufgabe
Business Workflow	Core::Class	Geschäftsvorfall
Proxy	Core::Class	Stellvertreter für ein in einer anderen Domäne definiertes Business Object
Event	Core::Class	Nachricht eines Events

Tabelle 7.2: Stereotypen für Business-Object-Diagramme

Powertyp-Konzept für Generalisierungsrelationen wird nicht benutzt.

Erweitert werden die UML-Klassendiagramme um sechs Stereotypen und zwei Tag Definitionen: Business Actor, Business Activity, Business Service und Business Workflow, sowie Proxy und Event als neue Stereotypen sowie ProducerOf und ConsumerOf als neue Tags. Tabelle 7.2 zeigt diese in das Profil aufgenommenen Stereotypen.

Ein Stereotyp „Business Entity“ wurde bewußt nicht aufgenommen. Denn die Business Entity Objects entsprechen am ehesten der ursprünglichen Bedeutung von Klassen in der traditionellen objektorientierten Modellierung. Mit Hilfe der Klassen wird dort zumindest in der Analyse das fachliche Objektmodell spezifiziert. Ein solches Vorgehen wird von den Entwicklern der UML bevorzugt:

> When capturing the extended semantics of a domain in the definition of a profile [...], modelers should be careful not to focus exclusively on defining stereotypes. In most cases a combination of stereotypes and predefined standard model elements will be most effective. [OMG99a]

Die gleiche Argumentation gilt für die Einführung von Stereotypen für die Assoziationen. Zwar bestehen – und dessen muß sich der Modellierer eines Business Object Modells immer bewußt sein – zwischen Assoziationen, je nachdem, welche Business Objects sie miteinander verbinden, beträchtliche konzeptionelle Unterschiede, sie werden aber auf Grund der Übersichtlichkeit immer in der gleichen Darstellung als Linie ohne zusätzliche Kennzeichnung gezeigt. Konzeptionell unterschiedliche Assoziation sind:

- Business Entity Object (BEO) $\longleftrightarrow$ BEO: Navigation, Aggregation und Life-Cycle (entspricht der üblichen Bedeutung von Assoziationen zwischen zwei Klassen in einem Klassendiagramm)
- Business Service Object (BSO) $\longleftrightarrow$ BEO: Manipulation, Kapselung
- Business Task Object (BTO)/Business Workflow Object (BWO) $\longleftrightarrow$ BEO: Manipulation, Life-Cycle
- BTO/BWO $\longleftrightarrow$ BSO: Nutzung
- BWO $\longleftrightarrow$ BTO: Aggregation, Life-Cycle
- BWO $\longleftrightarrow$ Business Actor Object (BAO): Nutzung
- BAO $\longleftrightarrow$ BEO: Rolle/Credentials (Autorisierung)
- BAO $\longleftrightarrow$ BTO: Aufgabenzuordung

Auf Stereotypen für diese unterschiedlichen Assoziationen wurde verzichtet, da bereits aufgrund der mit Stereotypen gekennzeichneten Assoziationspartner ersichtlich ist, welche Bedeutung die jeweilige Assoziation hat.

7.4.3 Abbildung auf das Metamodell

Im Gegensatz zu den Business-Use-Cases der Geschäftsvorfallsicht ist die Abbildung der Elemente des Business-Object-Diagramms umfangreicher, da nahezu das komplette Metamodell in der Struktursicht betrachtet wird. Die Abbildungen sind jedoch bis auf die Generalisierungsrelation nicht komplex, da das Business Object Metamodell und das UML-Metamodell beide MOF-konform sind und damit die Grundkonzepte in ähnlicher Weise implementieren.

Die folgenden Abbildungsbeziehungen sind kumulativ zu den bereits in Abschnitt 7.3.3 definierten Abbildungen.

- Core::Attribute $\mapsto$ Structure::Attribute
- Core::Operation $\mapsto$ Dynamics::Operation
- Core::Parameter $\mapsto$ Dynamics::Parameter
- Core::Class $\mapsto$ Structure::BusinessEntityObject

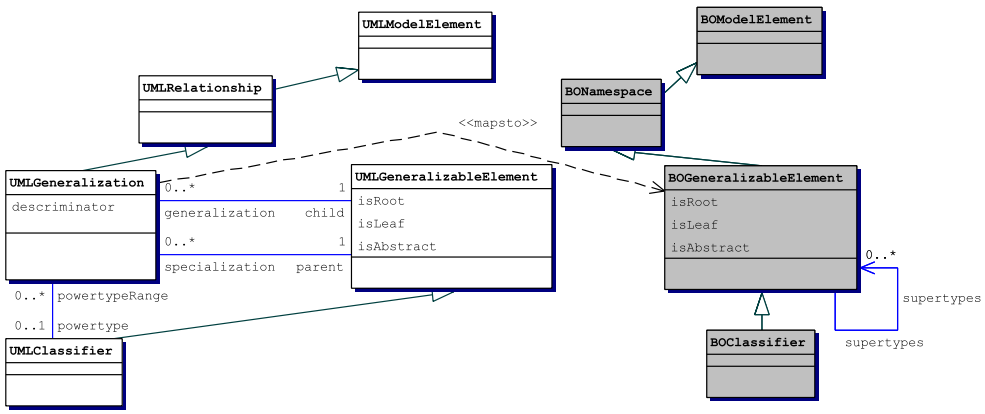

Abbildung 7.10: Mapping der Generalisierungsrelation

- Core::Generalization $\mapsto$ Base::GeneralizableElement.supertypes
- Core::DataType $\mapsto$ Structure::DataType

Die Generalisierungsrelation in UML wird durch zwei Referenzen (Listen) `generalization` und `specialization` des Metamodellelements `GeneralizableElement` auf ein zusätzliches Element `Generalization` dargestellt – im Gegensatz zum MOF-Konzept (und damit zum Business Object Metamodell), in dem die Vererbung durch die Referenz `supertypes` (Liste) von und zu dem Metamodellelement `GeneralizableElement` modelliert wird. Die Abbildung erfolgt, in dem jeweils der `child`-Eintrag der in Liste `generalization` geführten `Generalization`-Elemente zur Liste `supertypes` des entsprechenden Elements des Business Object Metamodells hinzugefügt wird.

7.4.4 Partitionierung

Das Beispiel in Abbildung 7.9 ist noch übersichtlich. In einem größeren Business Object Modell wird die Anzahl der darzustellenden Business Objects und Assoziationen in einem Business-Object-Diagramm zu umfangreich. Deshalb ist es sinnvoll, daß das Modellierungswerkzeug bestimmte Elemente vorübergehend auszublenden.

Dazu muß die Struktursicht durch eine geeignete Auswahl der Äquivalenz-Elemente partitioniert werden. Eine solche Partitionierung könnte sich zum Beispiel an den möglichen Assoziationen orientieren: Immer sichtbare Kernelemente

sind die Business Entity Objects und die BEO $\longleftrightarrow$ BEO Assoziation. Zusätzlich können alternativ oder gleichzeitig ausgewählt werden:

- BSOs: Durch Hinzunahme der BSO $\longleftrightarrow$ BEO Assoziationen
- BTOs und BWOs: Durch Hinzunahme der BTO/BWO $\longleftrightarrow$ BEO Assoziationen
- BAOs: Durch Hinzunahme der BAO $\longleftrightarrow$ BEO Assoziationen

7.5 Aufgabensicht

In der Geschäftsvorfallsicht werden die Vorgänge eines Geschäftsbereiches betrachtet. Dies erfolgt auf einer hohen Abstraktionsebene. Es werden nicht die einzelnen Aufgaben der Vorgänge und deren Zuordnung zu deren jeweiligen Bearbeiter modelliert. Dies ist Gegenstand der in diesem Abschnitt vorgestellten Aufgabensicht.

Die Aufgabensicht betrachtet die einzelnen Arbeitsaufgaben eines Geschäftsbereiches und weist diesen die zuständigen Prozeßbeteiligten zu. Unter Arbeitsaufgaben werden hierbei die nicht-arbeitsteiligen – also von genau einem Akteur zu bearbeitenden – Tätigkeiten verstanden, die eine diskrete, meßbare Manipulation des Zustandes eines Business Object Systems bewirken. Meßbar heißt, daß es Kriterien gibt, die entscheiden können, ob eine Aufgabe von einem Akteur abgeschlossen wurde.

Arbeitsaufgaben sind im Regelfall die nicht mehr weiter unterteilbaren Schritte eines Workflows. Dort ist eine Entscheidung, ob eine Aufgabe bereits erledigt wurde oder nicht unverzichtbar für die Steuerung des Kontrollflusses.

Die Aufgabensicht stellt diesen Kontrollfluß jedoch nicht dar. Sie bleibt auf einer mittleren Abstraktionsebene und beschreibt in einer White-Box-Sicht die einzelnen Bestandteile eines arbeitsteiligen Vorgangs. Der Kontrollfluß und damit die umfassende White-Box-Sicht eines Vorgangs wird erst in der Vorgangssicht modelliert, die im nächsten Abschnitt vorgestellt wird.

Die Aufgabensicht soll den Modellieren eines Business Object Modells die Möglichkeit geben, zunächst nur die Zuordnung von Arbeitsaufgabe zu den Akteuren zu betrachten, ohne kausale oder zeitliche Abhängigkeiten berücksichtigen zu müssen. Wichtig ist diese Sicht dann, wenn zum Beispiel ermittelt werden soll, welche Arbeitsaufgabe überhaupt einem Business Actor zugewiesen sind.

Zur Darstellung der Aufgabensicht werden wie in der Geschäftsvorfallsicht angepaßte UML-Use-Case-Diagramme, die sogenannten Business-Assignment-Diagramme, verwendet. Jedoch unterscheidet sich die Bedeutung der Business-Assignment-Diagramme von denen der Business-Use-Case-Diagramme aus Abschnitt 7.3 durch die Anwendung anderer Stereotypen auf die UML-Elemente.

7.5.1 Beispiel

Das folgende Beispiel führt das Modell aus Abschnitt 7.3.1 fort, indem es eine White-Box-Sicht des Geschäftsvorfalls „Konstruktionsmassnahme_Vorgang“ darstellt. Es wurde dort nur modelliert, daß es einen solchen Geschäftsvorfall gibt, in welchen Geschäftsbereich er stattfindet und von welchen Akteuren er bearbeitet wird.

In dem in Abbildung 7.11 gezeigten Business-Assignment-Diagramm werden nun die einzelnen Schritte des Geschäftsvorfalls in Form von Use-Case-Symbolen mit dem Stereotyp „«Business Task»“ und die Zuordnungen der Schritte zu den Bearbeitern dargestellt.

Die Schritte sind „MassnahmeAnlegen_Aufgabe“, in dem ein Auftraggeber den Geschäftsvorfall durch die Anlage einer neuen Konstruktionsmaßnahme initiiert. Ein Chef-Konstrukteur muß dann im Schritt „ArbeitsfolgeErstellen_Aufgabe“ die geeigneten Arbeitsfolgen für die Konstruktionsmaßnahme festlegen, die dann in ein oder mehreren Schritten „ArbeitsfolgeBearbeiten_Aufgabe“ von einem Konstrukteur bearbeitet werden. Nach Abschluß der Arbeitsaufgaben wird die Konstruktionsmaßnahme im Schritt „MassnahmePruefen_Aufgabe“ vom Chef-Konstrukteur geprüft und abschließend im Schritt „MassnahmeGenehmigen_Aufgabe“ von einem Freigabeverantwortlichen zur Produktion freigegeben.

Die Anordnung der fünf Aufgaben im Diagramm ist unerheblich und hat keine Bedeutung auf den Kontrollfluß zwischen ihnen. Die vier Business Actors entsprechen den Business Actors aus der Black-Box-Sicht des Geschäftsvorfalls. Zu Beachten ist auch, daß das dargestellte Diagramm nur ein Ausschnitt aller Arbeitsaufgaben eines Geschäftsbereichs ist. Das hier dargestellte Diagramm ist nur ein Ausschnitt und zeigt nur die Aufgaben des Geschäftsvorfalls „Konstruktionsmaßnahme_Vorgang“.

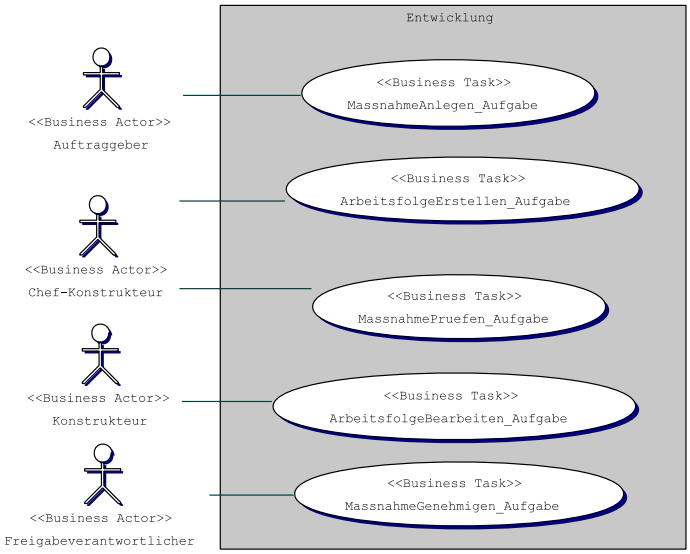

Abbildung 7.11: Beispiel eines Business-Assignment-Diagramms

7.5.2 UML-Erweiterungen

Die Business-Assignment-Diagramme basieren auf UML-Use-Case-Diagrammen. Wie schon bei den Business-Use-Case-Diagrammen aus Abschnitt 7.3 wird wegen der unklaren Semantik auf die Include-, Extend- und Generalisierungsbeziehung verzichtet. Das heißt, daß das Package „Use Cases“ für die Business-Assignment-Diagramme bis auf die Elemente „ExtensionPoint“, „Extend“ und „Include“ übernommen wird.

In Tabelle 7.3 werden die in das Profil zur Business Object Modellierung aufgenommenen Stereotypen zusammengefaßt.

Stereotyp	angewandt auf	Bedeutung
Business Actor	UseCases::Actor	Zuständiger Bearbeiter der Aufgabe
Business Task	UseCases::UseCase	Nicht-arbeitsteilige Aufgabe

Tabelle 7.3: Stereotypen für Business-Assignment-Diagramme

Die Relation „communicates“ zwischen den Business Actor und den Business Task Elementen wird auf die Assignment-Assoziation zwischen den Elementen BusinessActor und BusinessTask des Business Object Metamodells abgebildet und hat somit die Bedeutung „Aufgabe zugewiesen an“, statt „kommuniziert“ in den UML-Use-Case-Diagrammen.

7.5.3 Abbildung auf das Metamodell

Die Abbildung auf das Metamodell beschränkt sich, da kumulativ zu den bereits in Abschnitt 7.3.3 definierten Abbildung auf die Abbildung: UseCases::UseCase$\oplus$«Business Task» $\mapsto$ Structure::BusinessTaskObject.

7.6 Vorgangssicht

Die Vorgangssicht betrachtet die arbeitsteiligen Abläufe eines Unternehmens. Während in der Struktursicht die statische Struktur eines Business Object Systems hauptsächlich darstellt, spielt in der Vorgangssicht der Kontrollfluß der Workflows eine wesentliche Rolle.

Dabei ist die Vorgangssicht wie die Struktursicht eine Integration von Geschäftsvorfallsicht, Aufgabensicht und der noch vorzustellenden Organisationssicht mit zusätzlicher Angabe der kausalen und temporalen Abhängigkeiten zwischen Workflows, Sub-Workflows und Aktivitäten. Die entspricht der traditionellen Sicht der Ablauforganisation bei der Modellierung von Workflow-Modellen, ist allerdings nicht losgelöst von der übrigen Softwareentwicklung, sondern betrachtet nur Elemente des Business Object Metamodells.

Die Vorgangssicht beschäftigt sich jedoch nur zum Teil mit den Geschäftsprozessen einer Unternehmung. Wie in der Einleitung erwähnt, können aber müssen Geschäftsprozesse IT-unterstützt sein. In der Vorgangssicht interessieren aber nur diejenigen Geschäftsprozesse, die IT-unterstützt werden sollen und damit in das Business Object Modell abgebildet werden müssen.

Zur Modellierung dieser Vorgänge werden in der Vorgangssicht neben den eigentlichen Elementen eines Workflows, nämlich den Business Workflow Objects, den Business Task Objects und den Kontrollflußkonstrukten (Flow, Action) auch Business Entity Objects und Business Actor Object modelliert.

Dargestellt wird die Vorgangssicht durch Business-Workflow-Diagramme, eine Erweiterung der UML-Activity-Diagramme. Diese Diagramme sind ursprünglich zusätzlich zu den klassischen objektorientierten Modellierungstechniken in die UML aufgenommen worden gerade auch um Workflows zu modellieren. Jedoch müssen die Activity-Diagramme aufgrund ihrer problematischen Semantik und weitgehenden Isoliertheit zu den anderen Diagrammen entsprechend angepaßt und erweitert werden.

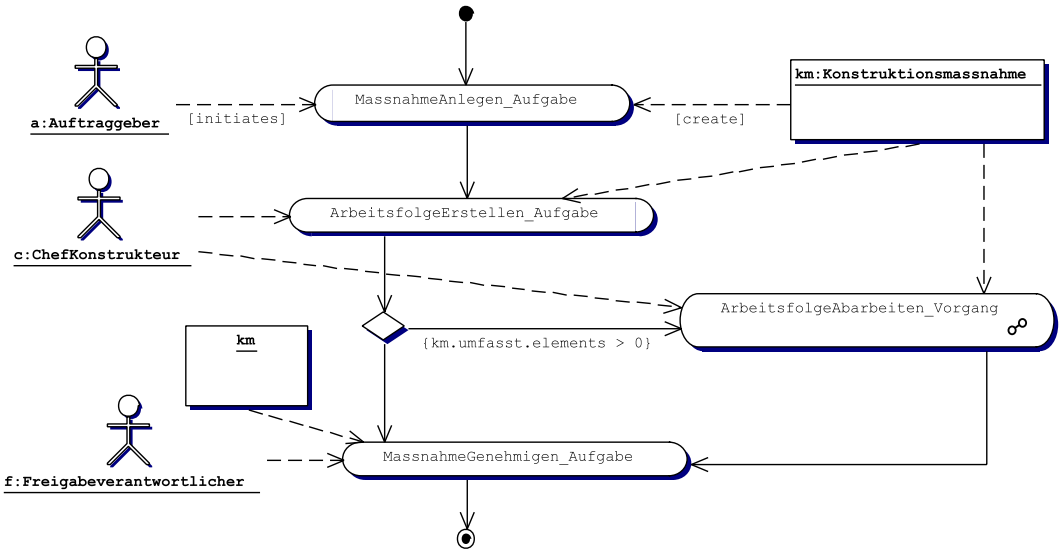

Abbildung 7.12: Beispiel eines Business-Workflow-Diagramms

7.6.1 Beispiel

Die in den Abbildungen 7.12 und 7.13 gezeigten Business-Workflow-Diagrammen führen das bisherige Beispiel weiter fort. Dargestellt wird nun die umfassende White-Box-Sicht des Geschäftsvorfalls „Konstruktionsmassnahme_Vorgang“. Zu den Business Tasks und Business Actors aus der Aufgabensicht kommt nun noch der Kontrollfluß und die Zuordnung von Business Entity Objects zu den Aufgaben hinzu.

Der Workflow beginnt am sogenannten Startpunkt (ein ausgefüllter Kreis). Er zeigt an, welche Aufgabe – in diesem Fall die Aufgabe „MassnahmeAnlegen_Aufgabe“ – als erstes bearbeitet werden soll. Ist die zu dieser Aufgabe gehörige Zuweisung eines Business Actors – in diesem Fall „Auftraggeber“ – mit „[initiates]“ gekennzeichnet, dann kann der dargestellte Business Workflow als Top-Level-Workflow agieren. Das heißt, daß jeder Business Actor mit der Rolle „Auftraggeber“ eine neue Konstruktionsmaßnahme starten kann. Das heißt für das Business Object System, das kein konkreter Business Actor einer Rolle gefunden und der Aufgabe zugewiesen werden muß.

Auf der anderen Seite können den Aufgaben oder den Subworkflows auch Business Entity Objects zugewiesen werden. Zusammen mit dem Instanzennamen, in diesem Beispiel „km“ für eine konkrete Konstruktionsmaßnahme, werden so prozessrelevante Business Objects von Aufgabe zu Aufgabe weitergereicht. Ist

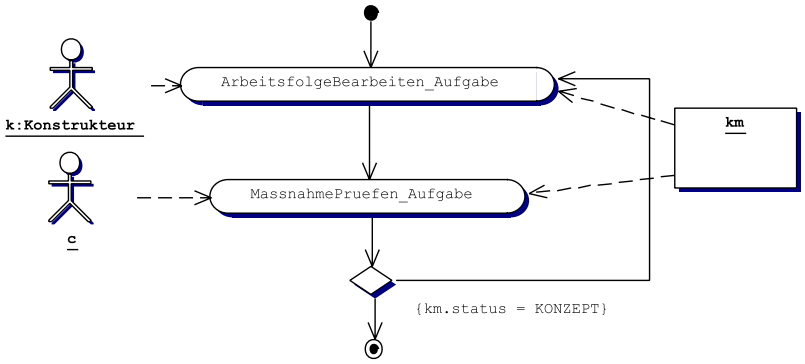

Abbildung 7.13: Weiteres Beispiel eines Business-Workflow-Diagramms

eine solche Zuweisung mit „[create]“ gekennzeichnet, so wird vor Beginn der jeweiligen Aufgabe eine neue Instanz des Business Entity Object erzeugt.

In diesem Beispiel füllt der Auftraggeber im ersten Schritt die notwendigen Felder der Konstruktionsmaßnahme „km“, also zum Beispiel die Begründung für die Maßnahme. Ist er mit dieser Aufgabe fertig, geht der Kontrollfluß an den nächsten Schritt „ArbeitsfolgeErstellen_Aufgabe“ über. Hier muß das System zuerst einen konkreten Business Actor, der die Rolle „ChefKonstrukteur“ hat, der Aufgabe zuweisen. Der Aufgabe wird ferner ein Business Object „Konstruktionsmaßnahme“ zugewiesen und zwar genau die Instanz, die durch „km“ gekennzeichnet wurde.

Als nächstes geht die Kontrolle auf ein Action Element über. In diesem Fall einer bedingten Verzweigung. Genau ein Ausgangsverlauf der beiden Ausgänge einer solchen Action sind mit einem booleschen Ausdruck gekennzeichnet. Aus Sicht des Metamodells ist ein solcher Ausdruck Teil einer Methode und wird damit in der Programmiersprache ausgedrückt, die in der Abbildung auf die Infrastruktur als Zielsprache ausgewählt wurde. In dieser Arbeit ist dies eine Teilmenge der Programmiersprache Java.

Der Action zugewiesen werden automatisch die Business Entity Objects, die auch der vorigen Aufgabe zugeordnet waren. Wenn in der Liste „umfasst“ der Arbeitsaufgaben der Konstruktionsmaßnahme Einträge vorhanden sind, wird geht der Kontrollfluß auf den Business Workflow „ArbeitsfolgenAbarbeiten_Vorgang“ über, der in Abbildung 7.13 dargestellt ist.

Die diesem Subworkflow zugewiesenen Business Actors „c“ (Chef-Konstrukteur) und Business Entities „km“ (Konstruktionsmaßnahme) werden den Aufgaben des Workflows dann zugewiesen, wenn dort ebenfalls mit „c“ oder „km“ gekennzeichnete Business Objects vorhanden sind und zwar die selben Instanzen. So ist zum

Beispiel gewährleistet, daß die Aufgabe „MassnahmePruefen_Aufgabe“ von der selben Person ausgeführt wird, die auch die Arbeitsfolgen in der Aufgabe „ArbeitsfolgenErstellen_Aufgabe“ definiert hat.

Ist der Vorgang „ArbeitsfolgeAbarbeiten_Vorgang“ angeschlossen, geht der Kontrollfluß wieder an den Workflow über, der den Subworkflow aufgerufen hat.

Nachdem auch die Aufgabe „MassnahmeGenehmigen_Aufgabe“ von einem Freigabeverantwortlichen bearbeitet wurde, zeigt der sogenannte Endpunkt (ausgefüllter Kreis mit einem umschließenden Kreis) bei dem Top-Level-Workflow den Abschluß des Workflows „Konstruktionsmaßnahme_Vorgang“ an.

Nicht gezeigt sind in diesem Beispiel die Fork und Join Elemente zur Kennzeichnung von parallelen Workflows. Auf sie wird noch im Abschnitt zur Abbildung auf die Infrastruktur (9.5) eingegangen.

7.6.2 UML-Erweiterungen

Die Modellierungskonzepte der UML-Activity-Diagramme sind umfassend genug, um die Vorgangssicht darzustellen. Nicht übernommen wurde die Möglichkeit, einen Workflow mithilfe der sogenannten Swimlanes zu partitionieren. Diesen Konzept eignet sich nur für Workflows, die wenig Aufgaben und Rollen enthalten. Zudem ist eine Zuweisung von konkreten Instanznamen zu den Rollennamen der Swimlanes nicht möglich, was eine Zuweisung ein und desselben konkreten Bearbeiters zu verschiedenen Aufgaben unmöglich macht.

Ebenfalls nicht übernommen wurde die Möglichkeit, mit den Event-Signalen (Signal) Kontrollflußübergänge zu triggern. Auf diese Elemente wurde verzichtet. Und schließlich wurde das Konzept des Objektflusses nicht mit der ursprünglichen Bedeutung übernommen, die problematisch bezüglich der gleichzeitigen Ausführung von parallelen Objekt- und Kontrollfluß, wie im Abschnitt

7.2.2 bereits diskutiert.

Stereotyp	angewandt auf	Bedeutung
Business Actor	ActivityGraphs::ObjectFlowState	Zuständiger Bearbeiter

Tabelle 7.4: Stereotyp für Business-Workflow-Diagramme

Erweitert werden mußte das UML-Activity-Diagramm um lediglich einen Stereotypen „Business Actor“ der auf die Darstellung eines Objektfluß-Zustandes

angewandt wird. Tabelle zeigt dieses Element des Profils.

7.6.3 Abbildung auf das Metamodell

Die Abbildung der Elemente des UML-Activity-Diagramms auf das Business Object Metamodell ist komplexer als bei den vorigen Diagrammtypen. Denn das Metamodell der Activity-Diagramme ist einerseits sehr isoliert von den sonstigen Metamodell-Elementen, andererseits haben die Entwickler der UML die Activity-Diagramme als Spezialfall von Zustandsautomaten aufgefaßt, was bedeutet, daß viele Metamodellelemente mit den Zustandsautomaten „geteilt“ werden.

Zunächst werden die einfachen Abbildungsbeziehungen vorgestellt, die sich kumulativ zu den bereits definierten Abbildungen verhalten:

- ActivityGraphs::ActionState $\mapsto$ Structure::BusinessTaskObject
- ActivityGraphs::SubactivityState $\mapsto$ Structure::BusinessWorkflowObject

Die komplexen Abbildungen betreffen die Transitionen, die Kontrollflußkonstrukte (Start- und Endpunkt, bedingte Verzweigung und Parallelausführung) sowie die Zuordnung von Business Actors und Business Entity Objects zu den Aufgaben und Subworkflows.

7.6.3.1 Transitionen

Die Abbildung der Transitionen erfolgt indem alle `Transition`-Elemente der UML auf `Flow`-Elemente des Business Object Metamodells abgebildet werden. Zusätzlich werden die Relationen `source` und `target` auf die entsprechenden Relationen `fromClassifier` und `toClassifier` abgebildet unter Beachtung der unten dargestellten Abbildung der Kontrollflußkonstrukte.

7.6.3.2 Kontrollflußkonstrukte

Die Kontrollflußkonstrukte Start- und Endpunkt, bedingte Verzweigung und Parallelausführungsbeginn und -ende sind in beiden Metamodellen enthalten, jedoch unterschiedlich modelliert. In Abbildung 7.15 ist der Unterschied dargestellt:

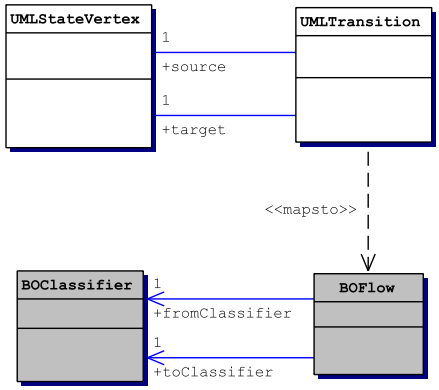

Abbildung 7.14: Mapping der Transitionen auf das Metamodell

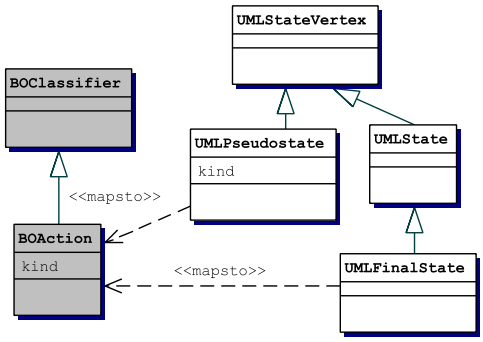

Abbildung 7.15: Mapping der Kontrollflußkonstrukte auf das Metamodell

- Startpunkt ist im UML-Metamodell ein `Pseudostate` mit dem `kind` „initial“. Dieses Element wird abgebildet auf ein `Action`-Element des Business Object Metamodell mit dem `kind` „start“.
- Endpunkt ist im UML-Metamodell als eigene Klasse `FinalState` modelliert. Dies wird abgebildet auf ein `Action`-Element mit dem `kind` „end“.
- Eine bedingte Verzweigung ist im UML-Metamodell ein `Pseudostate` mit dem `kind` „choice“. Daraus wird ein `Action`-Element mit dem `kind` „if“ abgeleitet.
- Bei den Elementen zur Modellierung eines Kontrollflußbeginns beziehungsweise -endes werden die `Pseudostate`-Elementen zu `Action`-Elementen, die `kind`-Werte „fork“ beziehungsweise „join“ sind gleich.

7.6.3.3 Zuordnungen

Für die Zuordnung von Business Actors und Business Entity Objects zu den Aufgaben, wurde bei den Business-Workflow-Diagrammen das Objektfluß-Konstrukt mit einer anderen Bedeutung verwendet.

Die Datenfluß-Zustände werden im UML-Metamodell durch das Element `ObjectFlowState` modelliert, eine Spezialisierung der Klasse `State`. Der Typ wird über die Relation `type` des `ObjectFlowState`-Elementes zu `Classifier`.

Im Falle des Stereotyps „Business Actor“, das auf einen `ObjectFlowState` angewandt ist (siehe vorigen Abschnitt), wird der mit `type` spezifizierte `Classifier` auf ein `BusinessActorObject` des Business Use Case Modells abgebildet und zusätzlich eine Assoziation zwischen dem `BusinessActorObject` und dem auf ein `BusinessTaskObject` beziehungsweise `BusinessWorkflowObject` abgebildeten `State` erzeugt, mit dem der `ObjectFlowState` verbundenen ist.

Genauso wird bei dem einfachen, nicht stereotypisierten `ObjectFlowState` vorgegangen. Allerdings wird hier der `Classifier` auf ein `BusinessEntityObject` abgebildet.

7.7 Organisationssicht

Ergänzend zur Vorgangssicht betrachtet die Organisationssicht die Aufbauorganisation eines Unternehmens. Die Aufbauorganisation beschreibt nach [PRW96] die Weisungs- und Organisationsbeziehungen zwischen den Organisationseinheiten eines Unternehmens. Auch werden den konkreten Akteuren einer Organisation in der Aufbauorganisation eine Menge von Rollen zugeordnet.

Letzteres ist besonders für die Ausführung von arbeitsteiligen Vorgängen wichtig. Die Zuordnung zwischen Business Actor und Business Task in der Ablauforganisation der Vorgangssicht beschreibt nur eine Rollenzuweisung. Welcher konkrete Akteur die Aufgabe dann bearbeitet, wird zur Laufzeit anhand der Informationen aus der Aufbauorganisation entschieden. Hier zeigt sich der enge Zusammenhang zwischen Vorgangs- und Organisationssicht im Business Object Modell.

Im Mittelpunkt der Organisationssicht stehen die Business Actor Objects, das heißt diejenigen Modellelemente, die eine Abbildung aller aktiven Elemente der Geschäftswelt darstellten, daß heißt diejenigen Elemente von denen ursprünglich eine Initiative zur Änderung beziehungsweise Abfrage des momentanen Zustands eines Business Object Systems ausgeht. Dies sind in der Regel Mitarbeiter des Unternehmens oder derer abstrakte Rollenrepräsentation, es können aber auch andere nicht-Business-Object-Systeme sein.

Zusätzlich werden die Organisationsstrukturen und ihre Beziehungen zu den Business Actors modelliert. In vielen Workflowmanagement-Systemen wird eine bestimmte Organisationsstruktur (Gruppe, Abteilung, Hauptabteilung, Bereich etc.) vorgegeben, die nur zum Teil an die Situation der jeweiligen Unternehmung anpaßbar ist. So gilt die oben genannte Klassifizierung zwar für den Großteil der Industrieunternehmen, Forschungseinrichtungen zum Beispiel haben jedoch eine ganz andere Struktur (Gruppe, Lehrstuhl, Institut, Fakultät). Deshalb ist ein Business Object Modell nicht von vornherein auf eine Struktur beschränkt. Organisationsstrukturen werden in der Organisationssicht durch Business Entity Objects und deren Assoziationen und Generalisierungen modelliert.

Umgesetzt wird die Organisationssicht angepaßten UML-Klassendiagrammen, den sogenannten Business-Organization-Diagrammen. Zu den bestehenden Elementen der UML-Klassendiagramme werden zusätzlich Stereotypen zur Kennzeichnung der Business Actor Objects verwendet und Constraints zur Zuweisung von Rolleneigenschaften.

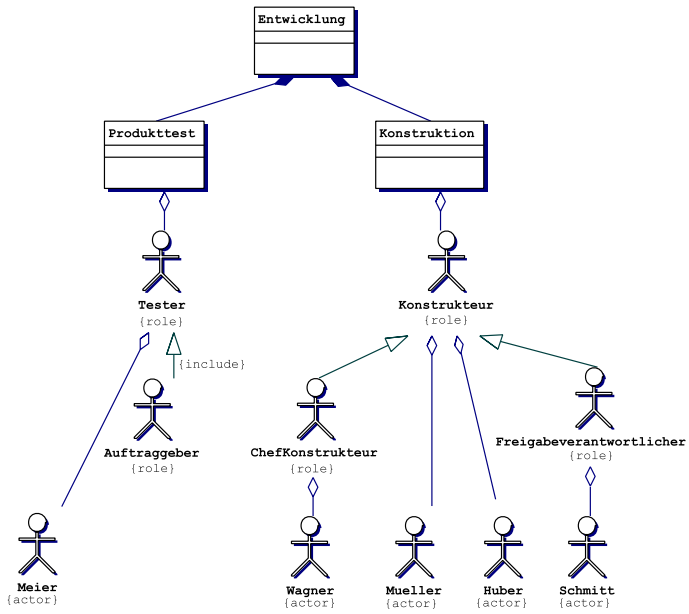

Abbildung 7.16: Beispiel eines Business-Organization-Diagramms

7.7.1 Beispiel

Die Aufbauorganisation des Beispiels zur Konstruktionsmaßnahme wird in Abbildung 7.16 dargestellt. Das Business-Organization-Diagramm zeigt die Organisationstruktur des Bereiches Entwicklung, in der es zwei Abteilungen gibt: Produkttest und Konstruktion.

Die Akteure dieser Abteilungen werden durch die Business Actor Stereotypen dargestellt. Einerseits die Rollen „Tester“, „Konstrukteur“, „Auftraggeber“, „ChefKonstrukteur“ und „Freigabeverantwortlicher“, gekennzeichnet mit dem Constraint „role“, andererseits die konkreten Akteure „Meier“, „Wagner“, „Mueller“, „Huber“ und „Schmitt“, gekennzeichnet mit dem Constraint „actor“.

Zwischen den Organisationseinheiten untereinander, den Organisationseinheiten und den Business Actors sowie zwischen den Business Actors untereinander gibt es verschiedene Beziehungen: Die Kompositionsbeziehung zwischen „Entwicklung“ und „Konstruktion“ gibt an, daß die Organisationseinheit „Entwicklung“ alle mit der Organisationseinheit „Konstruktion“ verknüpften Einheiten und Business Actors teilt.

Die Aggregationsbeziehung zwischen „Produkttest“ und „Tester“ gibt an, daß alle Rollen, die direkt oder indirekt von der Rolle „Tester“ abgeleitet sind und alle

Akteure die an die Rolle „Tester“ und deren abgeleiteten Rollen aggregiert sind, (nicht exklusiver) Teil der Organisationsstruktur sind.

Es gibt zwei Möglichkeiten Rollen zu generalisieren: Im Falle von „Konstrukteur“ und der Spezialisierung „ChefKonstrukteur“ bedeutet dies, das alle Akteure die direkt oder indirekt zur Rolle „ChefKonstrukteur“ aggregiert sind, auch die Rolle „Konstrukteur“ übernehmen können. Die an die Rolle „Konstrukteur“ und deren Oberklassen aggregierten Akteure – hier „Mueller“ und „Meier“ – werden aber nicht übernommen. Eine solche Übernahme von Akteuren wird durch das Constraint „include“ auf die Generalisierungsbeziehung ermöglicht, wie zwischen „Tester“ und „Auftraggeber“ gezeigt. In diesem Fall heißt das, daß „Meier“ auch die Rolle „Auftraggeber“ übernehmen kann.

Zwischen zwei Arten von Business Actors, den Rollen und Akteuren kann eine Aggregationsbeziehung wie im Fall der Akteure „Mueller“ und „Huber“ zur Rolle „Konstrukteur“ bestehen. Dies bedeutet, daß die Akteure die Rolle und deren Generalisierungen sowie alle Include-Spezialisierungen der Rolle im dem Business Object System einnehmen können. Dem Akteur „Wagner“ können demnach alle Aufgaben, die der Rolle „ChefKonstrukteur“ und „Konstrukteur“ zugeordnet sind, zugewiesen werden. Für den Vorgang „Konstruktionsmassnahme_Vorgang“ aus Abschnitt 7.6.1 heißt dies, daß „Wagner“ auch die Aufgabe „ArbeitsfolgeBearbeiten_Aufgabe“ zugewiesen werden könnte.

7.7.2 UML-Erweiterungen

Die Modellierungskonzepte der UML-Klassendiagramme als Basis der Business-Organization-Diagramme sind überausreichend. Irrelevant und nicht in das Profil übernommen werden: Interfaces (Core::Interface) und Abhängigkeits-Relationen (Core::Dependency, Core::Binding, Core::Usage und Core::Permission). Das Powertyp-Konzept für Generalisierungsrelationen wird ebenfalls nicht benutzt.

Erweitert werden die UML-Klassendiagramme um drei Constraints, die dem Stereotyp „«Business Actor»“ und der Generalisierungsrelation zwischen Business Actors hinzugefügt werden können. Der Stereotyp „«Business Actor»“ muß nicht erneut definiert werden, da er bereits in der Struktursicht (Abschnitt 7.4) zum Profil hinzugefügt wurde.

Tabelle 7.5 zeigt diese in das Profil aufgenommenen Constraints.

Constraint	angewandt auf	Bedeutung
actor	Core::Class⊕«Business Actor»	BAO ist ein Akteur
role	Core::Class⊕«Business Actor»	BAO ist eine Rolle
include	Core::Generalizaton	Akteure der Generalisierung werden übernommen

Tabelle 7.5: Constraints für Business-Organization-Diagramme

7.7.3 Abbildung auf das Metamodell

Die Abbildung auf das Metamodell ist bereits durch die in Abschnitt 7.4.3 definierten Abbildungen der Struktursicht abgedeckt.

7.8 Integration der Sichten

Ziel der in den vorigen Abschnitten vorgestellten Sichten auf ein Business Object System ist es, dem Modellierer die Möglichkeit zu geben, die verschiedenen Aspekte des Systems jeweils optimal spezifizieren zu können. Deshalb wurden in den fünf vorgestellten Sichten auch jeweils andere Diagrammtypen und damit Modellierungskonzepte und Darstellungsformen gewählt.

Durch die Abbildung der Modellelemente der gewählten Diagrammtypen auf das Business Object Metamodell und der Einschränkung der Modellierungskonzepte der Diagramme auf die Möglichkeiten des Metamodells, wurde eine Integration der Sichten erreicht.

In diesem Anschnitt werden die Zusammenhänge zwischen den Diagrammen der fünf Sichten dargestellt. Dies beschränkt sich jedoch nur auf die konzeptuelle Integration. Im nächsten Kapitel zum Vorgehensmodell werden dann Strategien vorgestellt, die Sichten auf ein Business Object Modell in Abhängigkeit zu bestimmten Phasen der Softwareentwicklung zu nutzen. Auch Übergänge oder Ableitungen von Diagrammtyp zu Diagrammtyp werden dort besprochen.

In Abbildung 7.17 wird gezeigt, wie die Diagramme des Beispiels „Konstruktionsmaßnahme“ zusammenhängen. Jeder graue Pfeil bedeutet, daß im Diagramm ein und dasselbe Element des Business Objects dargestellt wird, nur aus anderer Sicht. Aus Übersichtsgründen wurden nur die Pfeile zwischen dem Business Actor Object „Auftraggeber“, dem Business Entity Object „Konstruktionsmaßnahme“, der Domain „Entwicklung“, des Business Task Objects „Arbeitsauf-

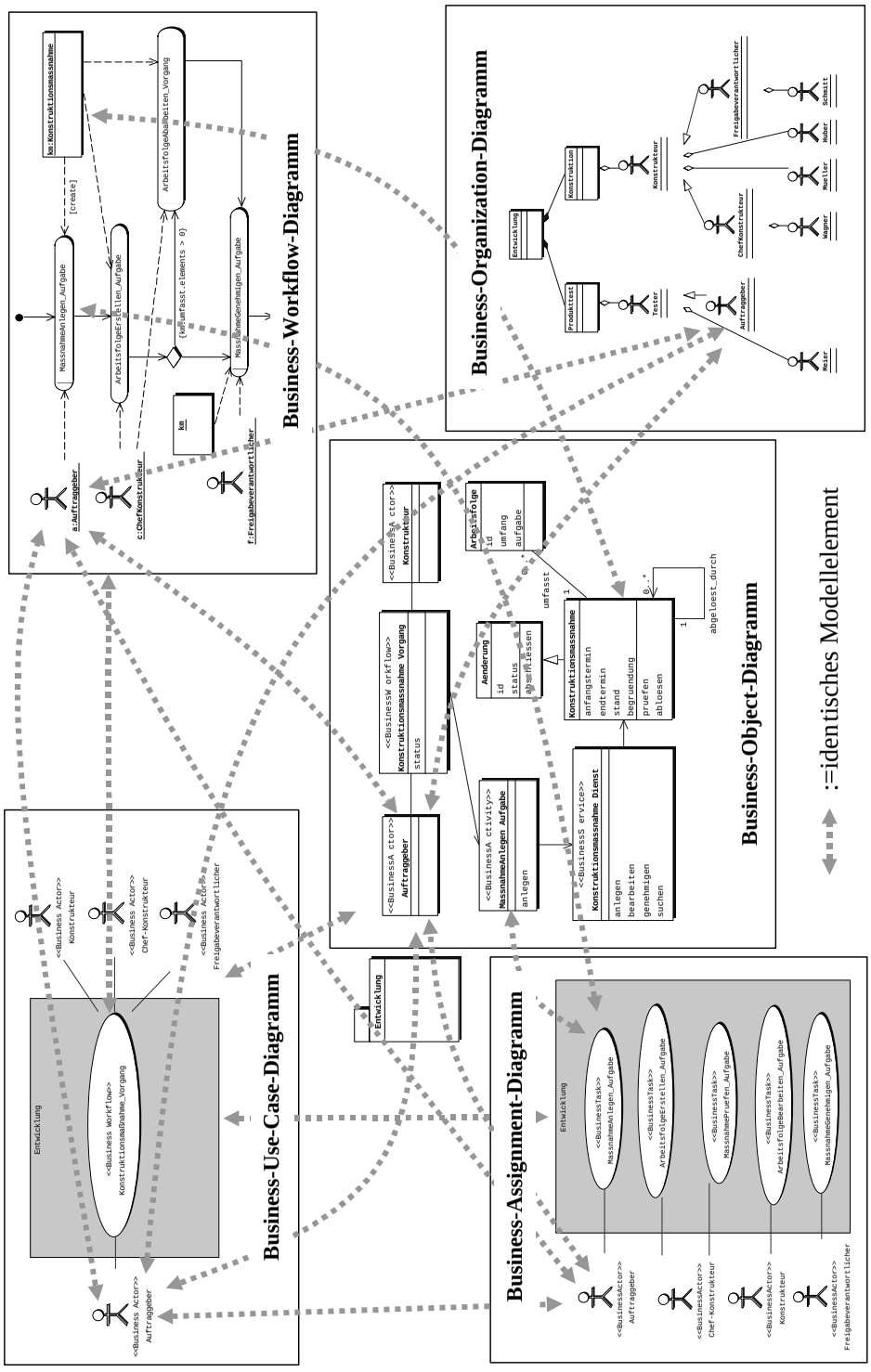

Abbildung 7.17: Integration der Diagramme zu „Konstruktionsmaßnahme“

gabeBearbeiten_Aufgabe“ und des Business Workflow Objects „Konstruktionsmaßnahme_Vorgang“ eingezeichnet.

Die generellen Konzepte des Metamodells, die mit den fünf verschiedenen Diagrammarten modelliert werden können, sind in Tabelle 7.6 zusammengefaßt. Dabei gilt für die Integration: Befindet sich in zwei Diagrammen verschiedener Sichten ein Modellelement gleichen Typs und gleichen Namens, dann zeigen die Diagramme in diesem Punkt das selbe Element des Business Object Modells und hängen somit logisch voneinander ab.

Diese Abhängigkeiten bedingen bei großen Business Object Modellen eine geeignete Werkzeugunterstützung zur Darstellung, Modellierung und Verwaltung des Modells. Gerade weil die Diagramme voneinander abhängen ist es notwendig, daß der Modellierer von Werkzeug eine Unterstützung erhält, die

- eine Navigationsmöglichkeit zwischen den Sichten,
- eine Konsistenzprüfung der Diagrammabhängigkeiten und
- eine Partitionierung des Modells

vorsieht. Ein Konzept für eine solche Unterstützung wird im Kapitel 9 vorgestellt. Das in dieser Arbeit entwickelte Profil ist eine Basis des Konzepts.

7.9 Zusammenfassung

Prinzipiell wäre es möglich, ein Business Object System allein in der Sprache des im vorigen Kapitel vorgestellten Metamodells entwerfen. Bei einem größeren Modell stößt ein solches Vorgehen schnell an die Grenzen der Machbarkeit für die Modellierer. In diesem Kapitel wurden deshalb graphische Modellierungstechniken vorgestellt, die die Möglichkeit bieten, verschiedene Sichten auf ein Business Object System zu erhalten. Dabei sind die daraus resultierenden Modelle ein vollständiges Abbild des modellierenden Systems, denn sie decken alle Systemkonzepte des Business Object Metamodells ab.

Um die Bedeutung der Beschreibungstechniken für die Modellierung zu klären wurden zunächst fünf unterschiedliche Sichten auf das Modell vorgestellt. Diese Sichten gewichten die Aspekte eines Business Object Modells unterschiedlich stark, wobei sie sich gegenseitig ergänzen. Zu jeder Sicht wurde denn im nächsten Schritt eine geeignete Beschreibungstechnik aus der UML ausgewählt und im

Diagrammart	Metamodell-Elemente
Business-Use-Case	Business Workflow Object Business Actor Object Association BAO - BWO Domain
Business-Assignment	Business Task Object Business Actor Object Association BAO - BTO Domain
Business-Workflow	Business Entity Object Business Task Object Business Workflow Object Business Actor Object Association BAO - BTO Association BEO - BTO Flow Action
Business-Object	Business Entity Object Business Service Object Business Task Object Business Workflow Object Business Actor Object Association BEO - BEO Association BEO - BSO Association BSO - BTO Association BTO - BWO Association BAO - BTO Association BAO - BWO Generalization BEO Generalization BSO Domain Event Attribute Operation
Business-Organization	Business Entity Object Business Actor Object Association BAO - BEO Association BAO - BAO Generalization BAO Attribute Operation

Tabelle 7.6: Metamodell-Elemente der Diagrammarten

Sicht	Diagramm	basiert auf
Geschäftsvorfallsicht	Business-Use-Case	Use-Case-Diagramm
Struktursicht	Business-Object	Klassendiagramm
Aufgabensicht	Business-Assignment	Use-Case-Diagramm
Vorgangssicht	Business-Workflow	Aktivitätsdiagramm
Organisationssicht	Business-Organization	Klassendiagramm

Tabelle 7.7: Übersicht über die Sichten und Diagrammarten

Rahmen eins UML-Profils an die Anforderungen zur Modellierung von Business Object Systemen angepaßt.

In Tabelle 7.7 sind alle Sichten, die jeweiligen Diagrammarten und die verwendeten UML-Diagramme zusammengefaßt.

Alle Diagrammarten wurden auf das gemeinsame Business Object Metamodell abgebildet und die Bedeutung der Modellelemente auf struktureller Ebene festgelegt. Durch die Abbildung auf das Metamodell bieten die entwickelten Diagrammarten für die Entwickler eines Business Object Systems einen integrierten und vollständigen Satz an Beschreibungstechniken.

Die Diagramme an sich geben dem Entwickler keinen Hinweis, in welchen Schritten er ein Business Object System entwickeln soll und wie dabei die unterschiedlichen Sichten genutzt werden können um ein Modell systematisch zu spezifizieren. Im nächsten Kapitel wird deshalb ein Vorgehensmodell vorgestellt, das auf Basis des Business Object Metamodells und der in diesem Kapitel entwickelten Beschreibungstechniken die Schritte zur Entwicklung eines Business Object Systems vorstellt.

Kapitel 8

Vorgehensmodell

Gängige Vorgehensmodelle (siehe Definition 2.20 (Vorgehensmodell)), wie das V-Modell [BD92] oder der Unified Process [JBR98] sind ausgerichtet auf die klassische Entwicklung *eines* vollständigen Systems innerhalb *eines* Projektes mit einer Vielzahl von Produkten, Sichten und Modellen zur Beschreibung aller Aspekte der Systementwicklung.

Die Entwicklung eines Business Object Systems entspricht nicht diesem klassischen Vorgehen. Ein Business Object System wird nicht als einzelnes System einmal entwickelt, sondern befindet sich – wie von ihm abgebildete Geschäftswelt – in einer kontinuierlichen Entwicklung. Deshalb sind die derzeitigen Vorgehensmodelle nur bedingt einsetzbar und müssen an die Entwicklung von unternehmensweiten Anwendungen auf Basis von Business Objects angepaßt werden. Dazu gehört vorrangig die Definition von Tätigkeiten und Produkten auf Basis des entwickelten Business Object Metamodells und der von diesem abgeleiteten Beschreibungstechniken.

In diesem Kapitel wird ein solches Vorgehensmodell vorgestellt. Die wesentlichen Ergebnisse des Kapitels sind:

- Ein eigenständiges Vorgehensmodell zur Entwicklung offener, unternehmensweiter Systeme auf Basis des Business Object Metamodells.
- Rollen und Organisationsformen speziell für die Anforderungen einer kontinuierliche Entwicklung und Weiterentwicklung von Business Object Systemen.
- Eine Variante eines Standard-Vorgehensmodells angepaßt an die Anforde-

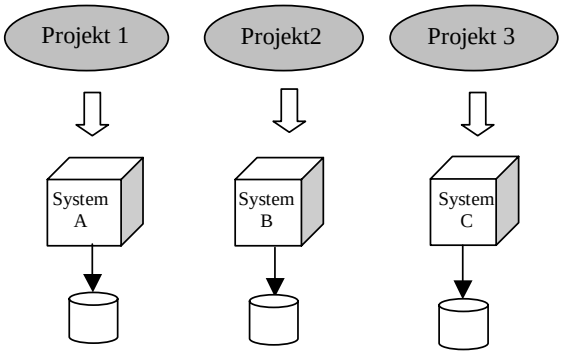

Abbildung 8.1: Anwendungsszenario derzeitiger Entwicklungsmethoden

rungen an die Entwicklung von Business Object Anwendungen.

Im folgenden Abschnitt werden zunächst die Besonderheiten eines unternehmensweiten Vorgehens zur Erstellung von Business Object Anwendungen dargestellt. Da das vorgestellte Vorgehensmodell eine Ergänzung des Unified Process ist, wird danach der Unified Process kurz vorgestellt. Im Mittelpunkt des Kapitels stehen die beiden hauptsächlichen Entwicklungsszenarien bei der unternehmensweiten Erstellung von Business Object Anwendungen, nämlich die Entwicklung und Evolution des Business Object Systems und die Entwicklung von Anwendungen oder die Anbindung von anderen Systemen als Sichten auf dieses System. Abschließend werden die Ergebnisse des Kapitels diskutiert und zusammengefaßt.

8.1 Besonderheiten eines unternehmensweiten Vorgehens

Die derzeit verfügbaren Entwicklungsmethoden haben in der Regel drei Eigenschaften, die nicht wünschenswert sind für den speziellen Fall eines unternehmensweiten Vorgehens zur Entwicklung von Business Object Systemen:

Universal Sie decken mit ihrem Modell eine große Bandbreite an möglichen Entwicklungsszenarien ab: Von der Entwicklung von eingebetteten Systemen, betriebliche Informationssystemen, über Anwendungsprogrammen bis hin zur Entwicklung von Werkzeugen zur Softwareentwicklung. Der Nachteil dabei ist, daß die Modelle dadurch sehr vage und redundant sind. Für die

genaue Definition eines Vorgehenmodells für spezielle Szenarien müssen sie zunächst – falls überhaupt möglich – angepaßt werden, so wie es mit dem Unified Process in dieser Arbeit geschieht.

Umfassend Dadurch, daß die Entwicklungsmethoden möglichst für viele Szenarien einsetzbar seinen sollen, besteht das Vorgehensmodell aus einer Vielzahl von Produkten, Sichten und Modellen, damit die Beschreibung möglichst vieler Aspekte der unterschiedlichen Softwaretypen abgedeckt ist. Dies hat den Nachteil, das die Produkte, Sichten und Modelle oftmals nur unzureichend miteinander verzahnt sind. Das Angebot von verschiedenen Schritten zu einem fertigen Produkt und mehreren, nicht integrierten Beschreibungstechniken für den gleichen Aspekt eines Systems sind dann für den Entwickler nicht immer hilfreich.

Projektbezogen Ausgerichtet sind die Vorgehensmodelle auf den Fall, ein komplettes System im Rahmen eines Projektes zu entwickeln. Dementsprechend sind auch die Rollen der Beteiligten des Modells überwiegend projektbezogen. Im Grunde genommen standen bei der Entwicklung der derzeitigen Entwicklungsmethoden diejenigen Unternehmen als Anwender im Fokus, die Software für andere Unternehmen herstellen. Dort ist natürlich jedes Projekt neu und im Regelfall unabhängig von Projekten mit anderen Kunden. Werden aber Systeme entwickelt, die von einem festen Personenkreis fortlaufend weiterentwickelt werden und die die Entwicklung anderer Systeme beeinflussen beziehungsweise davon beeinflußt werden, so reichen die projektbezogenen Rollen und Produkte nicht aus.

In Abbildung 8.1 wird dieses Anwendungszenario derzeitiger Entwicklungsmethoden dargestellt: Unabhängige Projekte entwickeln eigenständige, komplette Softwaresysteme. Komplett bedeutet zum Beispiel bei Informationssystemen, daß von der Dialoggestaltung, über die fachlichen Komponenten bis zur Datenhaltung alle Aspekte eines Systems analysiert, entworfen und implementiert werden.

Demgegenüber stehen die Besonderheiten eines unternehmensweiten Vorgehens auf Basis von Business Objects. Hauptsächlich ergeben sie sich aus der speziellen Architektur von Business Object Systemen, die das Business Object Modell mit der Abbildung der Geschäftswelt auf der einen Seite und unabhängig davon die Sichten – die Business Object Anwendungen – auf dieses Modell auf der anderen Seite als Produkte beinhalten.

Abbildung 8.2 zeigt das Anwendungsszenario „Entwicklung eines Business Object Systems“. Das unternehmensweite System setzt sich zusammen aus dem Business Object Modell und den Anwendungen auf diesem Modell. Ein solches

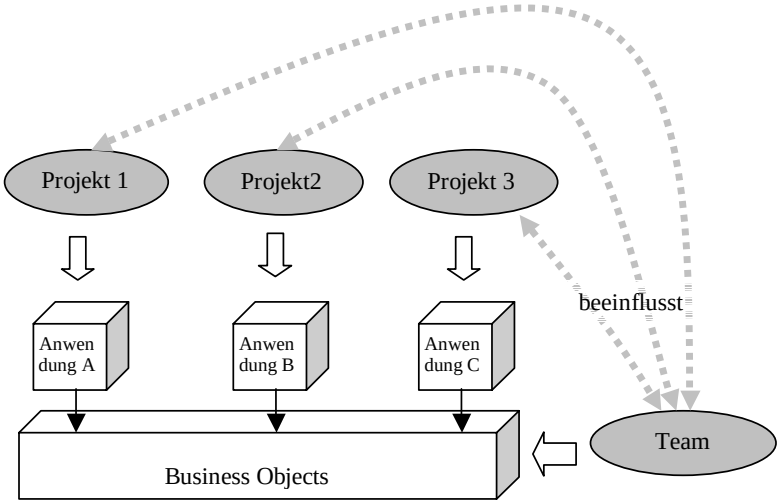

Abbildung 8.2: Anwendungsszenario Business Object System

System kann nicht innerhalb eines Projektes entwickelt werden. Denn das Modell ist zum Einen nach den Kriterien aus Kapitel 3 unabhängig von den Anwendungen. Zum Anderen ist das Modell als Basis für die gesamte Anwendungslandschaft eines Unternehmens gedacht und hat dadurch einen anderen Lebenszyklus als die Anwendungen. Zudem muß das Modell je nach dem, welche Anwendungen hinzukommen, fortlaufend weiterentwickelt werden. Es hat, und das ist ja gerade ein Meilenstein eines traditionellen Entwicklungsprojektes, keine Auslieferung bzw. kein klar definiertes Projektende.

Deshalb wurden die Beteiligten, die dem Business Object Modell in Abbildung 8.2 zugeordnet sind, auch als „Team“ und nicht als „Projekt“ gekennzeichnet.

Die Projekte zur Entwicklung von Business Object Anwendungen sind zwar in den meisten Fällen unabhängig voneinander, sie werden jedoch beeinflußt von der (Weiter-)Entwicklung des Business Object Modells, auf dem sie aufsetzen. Hier müssen im Entwicklungsprozeß Schnittstellen definiert sein, damit Projekt und Team die erforderlichen, gemeinsam genutzten Entwicklungsprodukte in einer systematischen Art und Weise austauschen und nutzen können.

Ein geeignetes Vorgehensmodell muß also Schritte, Produkte und Rollen vorsehen, die eine solch notwendige Koordination und Kooperation der Business Object Modellentwicklung und der Anwendungsentwicklung möglich machen. Auch die Anbindung von nicht-Business-Object-Systemen an das Modell muß dabei berücksichtigt werden.

Weitere Besonderheiten ergeben sich aus den Eigenschaften des Business Object Modells selbst. Wenn alle in Kapitel 3 aufgeführten Kriterien berücksichtigt werden, sind zur Modellierung der Business Objects nur ein Teil der Aspekte zu betrachten, die bei der Entwicklung von kompletten Softwaresystemen eine Rolle spielen. So ist zum Beispiel die Entwicklung technischer Komponenten und der Entwurf und die Realisierung der Benutzerschnittstelle vom Business Object Modell abgekoppelt. Dementsprechend kann sich ein angepaßtes Vorgehensmodell auf die im vorigen Kapitel vorgestellten, logischen Sichten beschränken. Diese Beschränkung hat den Vorteil, daß die zu erstellenden Produkte im Entwicklungsprozeß besser integriert und aufeinander abgebildet werden können.

Und schließlich ist die nahtlose Integration von arbeitsteiligen Vorgängen in das Business Object Modell ein besonderes Merkmal des unternehmensweiten Vorgehens. Es erfordert eine Schnittstelle zum Business Process Reengineering, um die IT-unterstützten Geschäftsprozesse identifizieren und in das Modell übernehmen zu können. Diese Schnittstelle zwischen Business-Engineering und Software-Engineering wird in vielen Vorgehensmodellen zur Softwareentwicklung vernachlässigt oder nicht vorgesehen.

Die genannten Besonderheiten zeigen, daß ohne Anpassung keines der klassischen Vorgehensmodelle geeignet ist, ein unternehmensweites Vorgehen auf Basis des vorgestellten Business Object Konzepts zu realisieren. In den folgenden Abschnitten wird eine solche Anpassung des Unified Process vorgestellt. Diese Anpassung wird für die Entwicklung des Business Object Modells und für die Entwicklung der Business Object Anwendungen getrennt definiert und in den Abschnitten 8.3 und 8.4 beschrieben. Im nächsten Abschnitt wird zunächst der Unified Process kurz vorgestellt.

8.2 Unified Process

Der Unified Process [JBR98] ist ein relativ neues Vorgehensmodell, basiert aber auf einer bereits erprobten Software-Entwicklungsmethode, der Objectory Methode (OOSE) von Jacobson [JCJO92].

Ausgewählt für die Basis eines unternehmensweites Vorgehens zur Entwicklung von Business Object Systemen wurde der Unified Process, weil er einerseits ein modulares, erweiterbares Grundkonzept bestehend aus Workflows, Phasen und Iterationen hat und weil er andererseits vor allem auf der Verwendung der UML als Modellierungssprache der Entwicklungsprodukte basiert.

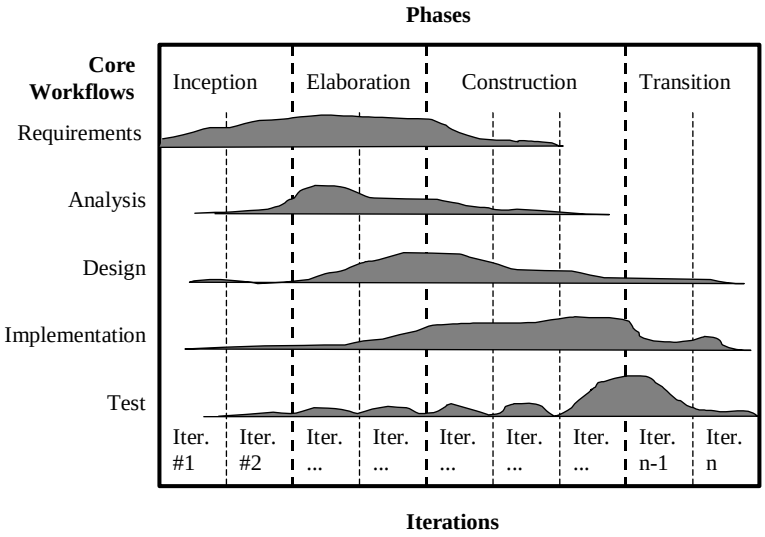

Abbildung 8.3: Grundgerüst des Unified Process (aus [JBR98])

8.2.1 Grundlagen

Der Unified Process wurde von vorherein mit dem Ziel entwickelt, auf unterschiedliche Entwicklungsszenarien anpaßbar zu sein.

> The Unified Process is more than a single process; it is a generic process framework that can be specialized for a very large class of software systems, for different application areas, different types of organizations, different competence levels, and different project sizes. [JBR98]

Ein Beispiel für eine konkrete Instanz des Unified Process ist der Rational Unified Process (RUP) [Kru98]. Der RUP wurde mit dem Ziel entwickelt, ein geeignetes Vorgehensmodell für den Umgang mit den zahlreichen Rational Softwareentwicklungswerkzeugen zu bieten.

Die Hauptziele des Unified Process sind neben der Anpaßbarkeit eine Konzentration auf ein modellgestütztes, inkrementelles und iteratives Vorgehen bei der Softwareentwicklung. Die Anwendung des Unified Process soll schließlich in die Entwicklung einfach zu wartender, komponentenbasierter Softwaresysteme münden.

Das Grundgerüst des Unified Process Modells sind in Abbildung 8.3 gezeigt. Es setzt sich zusammen aus:

Zyklen Der Unified Process wiederholt sich in einer Serie von Entwicklungszyklen über den gesamten Lebenszyklus eines Softwareproduktes, also von der ersten Version über nachfolgende verbesserte oder erweiterte Versionen bis zur Ablösung durch ein anderes Produkt. Jeder Zyklus endet mit einem Produkt-Release. In Abbildung 8.3 ist *ein* solcher Zyklus dargestellt. Er besteht aus vier Phasen, die wieder unterteilt sind in mehrere Iterationen.

Phasen Ein Zyklus im Unified Process besteht aus den vier Phasen: Inception (Anfang), Elaboration (Ausarbeitung), Construction (Konstruktion) und Transition (Übergang). In der Phase „Inception“ werden die Anforderungen an das neue Produkt beziehungsweise an ein neues Release, das am Ende des Zyklus entstanden sein soll, erarbeitet. In der Phase „Elaboration“ werden die Anforderungen ausgearbeitet, das heißt sie werden analysiert, um festzustellen, welche Funktionen und Eigenschaften das fertige Produkt aufweisen muß und welche Architekturen eine sinnvolle Basis für das Design des Systems sind. In der Phase „Construction“ wird die Software erstellt bis zu dem Punkt, an dem alle in der Analysephase ermittelten Funktionen und Eigenschaften vom Produkt erfüllt werden. Das so entstandene Produkt wird als Beta-Release bezeichnet. Schließlich wird in der Phase „Transition“ das Beta-Release von erfahrenen Benutzern getestet und von den Konstrukteuren immer weiter von etwaigen Fehlern oder Problemen befreit, so daß am Schluß der Übergangsphase die Auslieferung des fertigen Produktes beziehungsweise des neuen Release erfolgen kann. Alle Phasen enden mit einem Meilenstein, der festlegt, welche Zwischen- und Endprodukte des Entwicklungsprozesses zu diesem Zeitpunkt fertig sein müssen.

Iterationen Jede Phase kann wiederum in mehrere Iterationen aufgeteilt werden, die eine Phase in mehrere inkrementelle Schritte aufteilen. Inkrementell bezieht sich hierbei auf die zu liefernden Ergebnisse für den Meilensteins einer Phase. Denn Ergebnis einer Iteration sind die sogenannten „Increments“, also Teile oder bis zu einem definierten Punkt fertiggestellte Produkte. Alle Increments zusammen ergeben dann die geforderten Produkte des Meilensteins einer Phase. Der Unified Process selbst definiert keine konkreten Increments. Diese müssen die Anwender des Unified Process vor jeder Phase definieren. So könnte festgelegt werden, daß in der Phase „Elaboration“ in einer ersten Iteration zunächst die Gliederung und der Glossar

des Pflichtenhefts erstellt wird, in der nächsten Iteration die einzelnen Kapitel des Pflichtenheftes geschrieben werden.

Workflows Die Workflows werden in jeder Phase und jeder Iteration durchlaufen. Sie werden Workflows genannt, weil sie einen arbeitsteiligen Vorgang beschreiben, in dem Aktivitäten des Entwicklungsprozessen auf Bearbeiter-Rollen aufgeteilt werden und die Abfolge der Aktivitäten festgelegt wird. Im Unified Process gibt es fünf Kern-Workflows: Requirements, Analysis, Design, Implementation und Test. Für jeden Workflow werden die beteiligten Rollen (zum Beispiel „System Analyst“ oder „User-Interface Designer“), die einzelnen Aktivitäten (zum Beispiel „Find Actors and Use Cases“ oder „Prototype User Interface“) und die in den Aktivitäten zu erstellenden Produkte (zum Beispiel „Use Case Model“ oder „GUI Prototype“) definiert. Eine genauere Beschreibung der Workflows erfolgt im nächsten Abschnitt.

Die Grundstruktur des Unified Process hebt sich von anderen Vorgehensmodellen dadurch ab, das die Phasen von den Inhalten getrennt wurden. In anderen Modellen werden durch die Phasen auch gleich die Inhalte definiert, wie zum Beispiel im klassischen Wasserfallmodell: Analyse $\Rightarrow$ Design $\Rightarrow$ Implementierung $\Rightarrow$ Test. Dies führt dazu, daß die Phasen überwiegend auf eine Tätigkeit festgelegt sind und die Notwendigkeit für eine Designänderung in der Implementierungsphase nur unzureichend bearbeitet werden kann oder ein (teurer) Rücksprung in die Design-Phase gemacht werden muß. Natürlich ist hier das Wasserfallmodell ein extremes Beispiel. Nachfolgende Modelle wie das V-Modell [BD92] oder das Spiralmodell [Boe88] haben versucht diese Nachteile zu vermeiden. Ihnen ist jedoch gemein, daß jeder Phase auch eine fachliche Tätigkeit (aus Sicht des Softwareentwicklungsprozesses) zugeordnet ist.

Der Unified Process erreicht mit dieser Aufteilung eine größere Skalierbarkeit im Sinne der Auswahl und Ausgestaltung der Workflows, daß heißt er kann besser an unterschiedliche Entwicklungszenarien angepaßt werden, weil es verhindert wird, daß die Workflows an eine Phase gebunden sind und somit nicht weggelassen oder im großen Umfang geändert werden könnten. Zum Beispiel wäre es schwierig, im V-Modell die Design-Phase zu überspringen, da alle nachfolgenden Phasen auf den Ergebnissen der Design-Phase basieren.

8.2.2 Workflows, Produkte und Rollen

Die Workflows des Unified Process und ihre Bestandteile sind diejenigen Elemente, über das das Prozeßframework angepaßt und erweitert werden kann.

Eine Definition eines Unified Process Workflows setzt sich zusammen aus Produkten, Rollen und Aktivitäten:

- Produkten (Artifacts), die im Workflow benötigt und produziert werden. Hierbei ist zu beachten, daß der Begriff „Produkt“ nicht nur für das Softwareprodukt, sondern für eine Vielzahl möglicher Ergebnisse eines Entwicklungsprozesses steht. Zum Beispiel das Benutzerhandbuch, ein Analysemodell oder ein Testfall.
- Rollen (Workers), die im Workflow von konkreten Personen eingenommen werden und im wesentlichen eine Beschreibung der jeweiligen speziellen Fähigkeit eines Mitglieds des Entwicklungsteams sind. Rollen sind zum Beispiel „Systemanalyst“ oder „Softwarearchitekt“.
- Aktivitäten (Activities), die den Workflow in einzelne, nicht weiter teilbare Tätigkeiten aufteilen. Für jede Aktivität wird definiert, was zu tun ist, welche Rollen für die Tätigkeit in Frage kommen und welche Produkte benötigt und produziert werden.

In Abbildung 8.4 ist beispielhaft der vordefinierte Workflow zur Anforderungsdefinition dargestellt. Auf der linken Seite sind die Rollen „System Analyst“, „Architect“, „Use-Case Specifier“ und „User-Interface Designer“ innerhalb von „Swimlanes“ modelliert. Die Aktivitäten sind durch Transitionen miteinander verbunden und definieren den Ablauf des Workflows. Nicht dargestellt sind hier die Produkte, die in der Beschreibung der einzelnen Aktivitäten modelliert werden.

Für weitere Einzelheiten zum Unified Process wird auf [JBR98] verwiesen. In den beiden folgenden Abschnitten wird der Unified Process als Basis für ein Vorgehensmodell zur Entwicklung eines Business Object Modells und zur Entwicklung von Business Object Anwendungen genutzt, daß heißt die vordefinierten Workflows angepaßt, erweitert oder neue Workflows hinzugefügt.

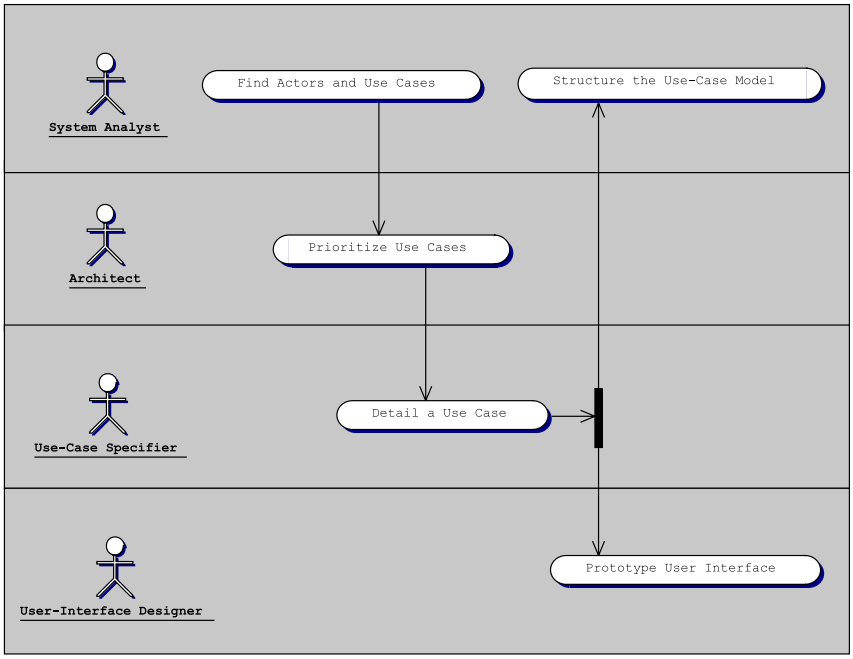

Abbildung 8.4: Workflow zur Anforderungsanalyse

8.3 Business Object Systementwicklung

Aufgrund der oben bereits aufgeführten Besonderheiten muß das Vorgehensmodell zur Entwicklung eines Business Objects Modell eine anderen Schwerpunkt haben, als ein Vorgehen zur Entwicklung eines eigenständigen Softwareprodukts. Die in den folgenden Abschnitten angepaßten Workflows des Unified Process tragen dem Rechnung, in dem neue Rollen, Aktivitäten und Produkte und die Beziehung zwischen ihnen eingeführt werden.

Grundsätzlich unterscheidet sich der Lebenszyklus eines Business Object Systems nicht wesentlich von dem eines traditionellen betrieblichen Informationssystems: In einem ersten Abschnitt wird das Business Object Modell, idealerweise nach einem Business Process Reengineering-Projekt aufgesetzt und die grundlegenden Business Objects des Geschäftsbereiches werden realisiert. Dann folgen Abschnitte, in dem das Business Object Modell durch Anforderungen von außen geändert oder erweitert werden muß.

Solche äußeren Anforderungen können sein:

- Neue Business Object Anwendung: Eine neue Anwendung soll auf Basis

des Business Object Modells arbeiten. Dabei wird es in den meisten Fällen so sein, daß die existierenden Business Objects geändert oder neue Business Objects hinzugefügt werden müssen.

- Änderung einer Business Object Anwendung: Hier gilt im Prinzip das selbe wie bei der Entwicklung einer neuen Anwendung. Es werden jedoch weniger häufig neue Business Objects hinzukommen, da sich üblicherweise eher Details im Geschäftsablauf der Anwendung ändern, aber nicht das Geschäftsfeld selbst.

- Änderung eines Geschäftsprozesses: Diese Anforderung kommt vom Business Process Reengineering. Werden dort neue Verfahren oder Vorgehensweisen eingeführt, so müssen diese umgehend in das Business Object Modell übernommen werden, damit die Abbildung von Geschäftswelt auf Business Object erhalten bleibt. Im Idealfall müssen hier nur der interne Ablauf von Business Workflow Objects geändert werden.

- Ankopplung eines nicht-Business-Object-Systems: Da diese Systeme nur lesend auf das Business Object Modell zugreifen sollten beziehungsweise als Business Actor Object nur Vorgänge anstoßen, sind die Auswirkungen auf das Modell in diesem Fall minimal.

Bei einem eigenständigen Informationssystem wird die Anzahl von Redesign- und Anpassungsmaßnahmen eher gering sein. Bei einer Lebensdauer von 10 Jahren ist mit ca. drei solchen Maßnahmen zu rechnen [Kau94]. Zu diesem Zweck werden Projekte aufgesetzt, die ähnlich ablaufen wie klassische Softwareentwicklungsprojekte. Im Gegensatz dazu werden die oben dargestellten Änderungs- und Erweiterungsanforderungen für ein Business Object Modell in einem großen Unternehmen unter Umständen wöchentlich gestellt.

Das bedeutet, daß nicht für jede Anforderung ein eigenes Projekt aufgesetzt werden kann. Sinnvoller ist es, für das Business Object Modell ein festes Team zu definieren, das aus Mitarbeitern besteht, die verschiedene spezialisierte Rollen zur Entwicklung und Wartung eines Business Object Systems einnehmen. Die Anforderungen müssen vom Team gebündelt werden und können dann in klassischer Projektform durchgeführt werden.

Solche Projekte durchlaufen dann einen Zyklus im Sinne des Unified Process. Die nachfolgend vorgestellten Workflows beschreiben die Rollen, Aktivitäten und Produkte, die in diesen Projekten eine Rolle spielen. Die Namen der Workflows, Rollen und Aktivitäten werden dabei in Englisch angegeben, um leichter schon

vorhandene oder neue Elemente im Vergleich mit dem generischen Unified Process zu erkennen. Ist ein Element übernommen worden, so wird dies explizit erwähnt, alle anderen Elemente sind speziell für ein unternehmensweites Vorgehen auf Basis von Business Objects entwickelt worden.

8.3.1 Workflow: Requirements

Der Requirements-Workflow beschäftigt sich mit der Umsetzung des Geschäftsmodells auf das Business Object Modell. Die Hauptaufgabe dieser Umsetzung ist es, diejenigen Teile des Geschäftsmodells zu identifizieren, die durch das Business Object System unterstützt werden sollen. Damit werden die Anforderungen festgelegt, die Festlegen, welche Business Objects neu in das System aufgenommen werden sollen oder welche Business Object geändert werden müssen.

8.3.1.1 Produkte

Dazu ist es zunächst einmal nicht notwendig alle Business Objects im Detail zu erfassen. Es genügt zunächst die hauptsächlichen Vorgänge und die jeweiligen Akteure festzulegen. Dazu eignen sich die in Abschnitt 7.3 vorgestellten Business-Use-Case Diagramme. Mit ihnen läßt sich das Modell der Geschäftsvorfälle in der Übersicht darstellen. Die zu erarbeitenden Produkte wären demnach das Modell der Geschäftsvorfälle, Business Actors und die (noch nicht konkretisierten) Business Workflows.

Weiterhin können in einem nächsten Schritt die Geschäftsvorfälle detailliert werden, in dem man die einzelnen Schritte der Geschäftsvorfälle modelliert. Dazu eignen sich die im Abschnitt 7.5 vorgestellten Business-Assignment-Diagramme, die ein Modell der zu bearbeitenden Aufgaben der Geschäftsvorfälle zu erstellen, ohne auf detaillierte, kausale oder zeitliche Abhängigkeiten zwischen den Aufgaben Rücksicht nehmen zu müssen. Produkte sind also das Modell der Aufgabenzuordnungen und die Business Tasks.

In jedem Schritt des Workflows werden zudem Testfälle erstellt. Sie dienen der Sicherstellung von funktionalen Eigenschaften nach Abschluß der Entwicklung des Business Object Systems.

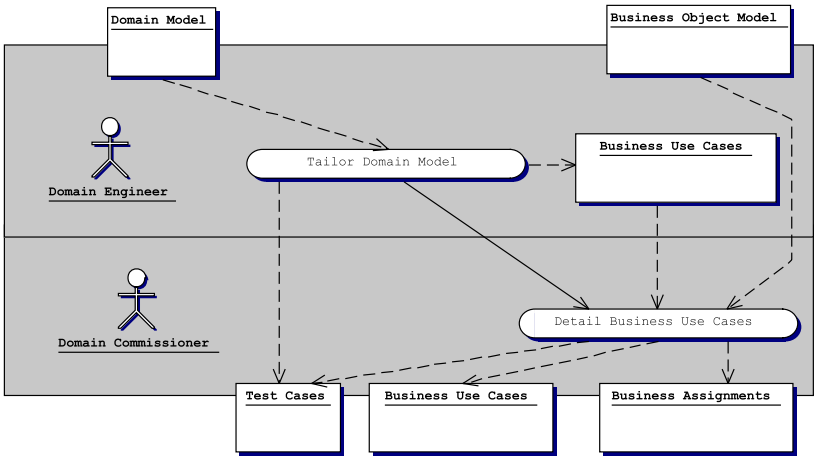

Abbildung 8.5: Requirements Workflow

8.3.1.2 Rollen

Der Requirements-Workflow wird durchgeführt vom Domain Engineer und dem Domain Commissioner. Der Domain Engineer ist ein Spezialist aus dem Bereich des Business Process Reengineering und entscheidet, welche Geschäftsvorfälle in das Business Object Modell übernommen werden sollen.

Der Domain Commissioner ist Mitglied des oben genannten Teams zur Entwicklung und Verwaltung des Business Object Systems. Für jeden Geschäftsbereich (Domain) gibt es einen dedizierten Beauftragten. Er kennt die Abhängigkeiten der Business Objects innerhalb seiner Domain und kümmert sich um die Einführung und Verknüpfung neuer Business Objects in „seinen“ Bereich. Das ist vor allen dazu notwendig, die Einführung redundanter Business Objects in das Modell zu verhindern. Außerdem fungiert der Domain Commissioner als Ansprechpartner für alle Projekte, die Business Object Anwendungen in der jeweiligen Domain planen.

8.3.1.3 Aktivitäten

Die Ergebnisse des Requirements-Workflows werden in zwei Aktivitäten produziert: „Tailor Domain Model“ und „Detail Business Use Cases“. In der Aktivität „Tailor Domain Model“ bestimmt der Domain Engineer diejenigen Teile eines umfassenden Business Reengineering Modells, die in das Business Object Modell übernommen werden sollen. Ergebnis ist das Geschäftsvorfallmodell in Form ei-

nes Business-Use-Case-Diagramms sowie Testfälle, die diese Geschäftsvorfälle betreffen.

Danach wird vom Domain Commissioner auf Basis des bestehenden Business Object Modells das Geschäftsvorfallmodell im Schritt „Detail Business Use Cases“ weiter ausgearbeitet und die einzelnen Business Task Objects und ihre Zuordnung zu den Business Actor Objects festgelegt. Ergebnis ist das detaillierte Geschäftsvorfallmodell und das Aufgabenmodell in Form eines Business-Assignment-Diagramms. Zudem werden Testfälle für die Aufgaben erstellt.

In Abbildung 8.4 ist der Requirements-Workflow in der Übersicht dargestellt. Das dargestellte Produkt „Domain Model“ ist kein explizites Element dieses Vorgehensmodells, sondern kommt aus einem separaten Prozeß zum Business Process Reengineering. Form und Darstellung sind dabei unerheblich. Das Business Object Modell steht für das bereits existierende Business Object System und stammt aus früheren Zyklen.

8.3.2 Workflow: Analysis

Im Analyse-Workflow werden die im vorigen Workflow ermittelten neuen oder geänderten Geschäftsprozesse analysiert. Das heißt, daß die Business Workflow Objects, Business Actor Objects und Business Task Objects auf eine Business Object Architektur abgebildet werden. Architektur im Sinne der verschiedenen Beziehungen und Abhängigkeiten zwischen den Business Objects. Ziel des Workflows ist es, die neue Struktur des Business Object Modells mit den zusätzlichen oder geänderten Geschäftsvorfällen zu erhalten.

8.3.2.1 Produkte

Benötigt werden im Analysis-Workflow das Geschäftsvorfallmodell und das Business Object Modell des zu erweiternden Business Object Systems. Produziert werden in diesem Workflow ein erweitertes Business Object Modell in Form eines Business-Object-Diagramms und detailliertere Geschäftsvorfälle in Form von Business-Workflow-Diagrammen und Business-Assignment-Diagrammen.

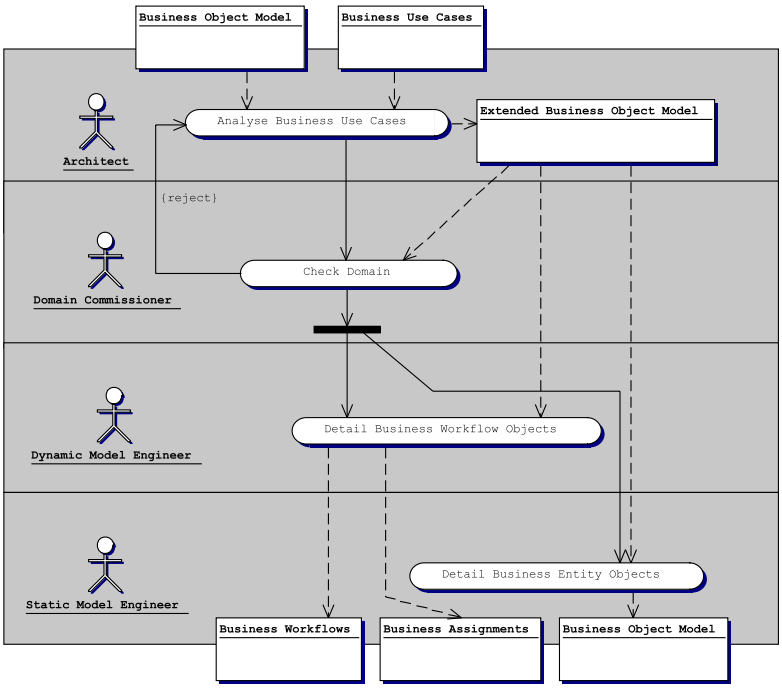

Abbildung 8.6: Analysis Workflow

8.3.2.2 Rollen

Neben der schon vom Requirements-Workflow bekannten Rolle „Domain Comissioner“, die in diesem Workflow die Erstellung eines erweiterten Business Object Modells kontrolliert, werden in diesem Workflow drei weiteren Rollen „Architect“, „Dynamic Model Engineer“ und „Static Model Engineer“ Aktivitäten zugeordnet.

Die Rolle „Architect“ wurde vom generischen Unified Process übernommen. Die Aufgabe des Architekten ist es, die grundlegende Architektur des Business Object Systems festzulegen. Mit grundlegender Architektur ist gemeint, welche Business Objects es überhaupt gibt und wie diese untereinander in Beziehung stehen.

Der „Dynamic Model Engineer“ ist Spezialist für arbeitsteilige Vorgänge, den Business Workflow Objects. Er weiß, wie Vorgänge am besten in einzelne Schritte und Subworkflows eingeteilt werden können. Der „Static Model Engineer“ ist demgegenüber Spezialist für die statischen Abhängigkeiten in einem Business Object System. Er weiß, wie Beziehungen der Elemente der Geschäftswelt in Assoziationen zwischen Business Objects umgesetzt werden.

8.3.2.3 Aktivitäten

Der komplette Analysis Workflow besteht aus vier Aktivitäten „Analyse Business Use Case“, „Check Domain“, „Detail Business Workflow Objects“ und „Detail Business Entity Object“.

Die Aktivität „Analyse Business Use Case“ wird vom Architekten bearbeitet. Er definiert aus den Business Object Modell des bisherigen Systems und den neuen Geschäftsvorfällen ein erweitertes Business Object Modell. Dies enthält unter Umständen neue Business Entity Objects und Business Service Objects und ihre Beziehungen zu den Business Workflow Objects der neuen Geschäftsvorfälle. Dieses erweiterte Business Object Modell wird als Business-Object-Diagramm dargestellt.

Als nächstes prüft der Domain Commissioner in der Aktivität „Check Domain“, ob die vom Architekten definierte Struktur nicht redundant innerhalb „seiner“ Domain ist. Falls Unstimmigkeiten vorhanden sind, wird die Aktivität „Analyse Business Use Case“ erneut durchgeführt.

Die beiden nächsten Aktivitäten können parallel ausgeführt werden. Zum Einen werden in der Aktivität „Detail Business Workflow Objects“ die hinzugekommenen Geschäftsvorfälle weiter verfeinert, in dem der Dynamic Model Engineer den einzelnen Schritten die benötigten Ressourcen (Business Entity Objects) zuweist, zum Anderen werden in der Aktivität „Detail Business Entity Objects“ die Assoziationen zwischen dem vom Architekten vorgegebenen Business Entity Objects hinzugefügt. Ergebnis der beiden Aktivitäten ist ein Business Object Modell, Business-Workflow-Diagramme und Business-Assignment-Diagramme.

Abbildung 8.6 zeigt den Analysis-Workflow in der Übersicht.

8.3.3 Workflow: Design

Im Design-Workflow werden die neuen Business Objects weiter detailliert und ihre Beziehungen untereinander komplettiert. Da das Framework der Arbeit auf eine stark modellbasierte Entwicklung von unternehmensweiten Systemen ausgerichtet ist, ist nach dem Abschluß des Designworkflows das Business Object Modell nahezu abgeschlossen. Implementiert müssen lediglich die Methoden der Business Object Operationen.

8.3.3.1 Produkte

In den Design-Workflow geht das im Workflow Analyse entwickelte, erweiterte Business Object Modell ein, sowie das Aufgabenmodell in Form von Business-Assignment-Diagrammen und das vorläufige Workflow-Modell in Form von Business-Workflow-Diagrammen.

Da wie oben schon dargestellt im Design-Workflow das nahezu komplette Business Object Modell fertiggestellt wird, sind auch die Produkte im Workflow umfangreich: Die Elemente eines Business Object Modells aus unterschiedlichen Sichtweisen in Business-Workflow-Diagrammen, Business-Object-Diagrammen und Business-Organisation-Diagrammen.

Zusätzlich wird ein Deployment-Plan erstellt, der sich bereits mit der Umsetzung des Modells auf die Business Object Infrastruktur beschäftigt und Aussagen trifft, welche Infrastrukturelemente benötigt werden und welche Mengengerüste für die neuen Business Objects erwartet werden.

Wie im Requirements-Workflow werden auch in diesem Workflow Testfälle erstellt. In diesem Fall aber im Gegensatz zu den Testfällen des Requirements sind dies im wesentlichen nicht-funktionale Tests zur Sicherstellung des Quality-of-Service des Business Object Systems. Zum Beispiel Testfälle zur Performance-Messung bei Business Entity Objects mit großem Mengengerüst.

8.3.3.2 Rollen

Die Aufgaben im Design-Workflow übernehmen die oben bereits eingeführten Rollen „Dynamic Model Engineer“ und „Static Model Engineer“. Neu ist die Rolle „Infrastructure Manager“.

Der „Dynamic Model Engineer“ komplettiert die Workflows, indem er Abläufe und Organisationsstrukturen hinzunimmt. Der „Static Model Engineer“ vervollständigt das Business Object Modell um Business Service Objects, Events sowie Attribute und Operationen.

Der „Infrastructure Manager“ ist zuständig für die Business Object Infrastruktur, auf die das Business Object Modell abgebildet wird. Er hat die Übersicht über die Möglichkeiten und Grenzen der Infrastruktur und kann entscheiden, ob ein bestimmtes Business Object Modell auf Basis der Infrastruktur realisierbar ist.

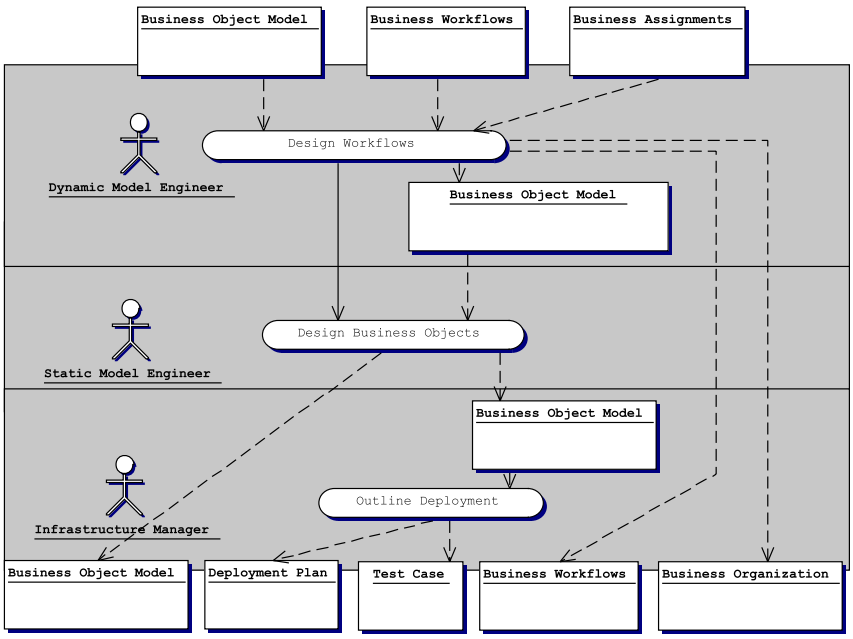

Abbildung 8.7: Design Workflow

8.3.3.3 Aktivitäten

Im Design-Workflow sind, wie in Abbildung 8.7 gezeigt, drei Aktivitäten vorgesehen: „Design Workflows“, „Design Business Objects“ und „Outline Deployment“.

In der Aktivität „Design Workflows“ werden vom „Dynamic Model Engineer“ die neuen oder geänderten Workflows weiter konkretisiert. Es werden die Transitionen zwischen den Workflowschritten hinzugefügt und das Organisationsmodell angepaßt. Dies geschieht in Form von Business-Workflow- und Business-Organization-Diagrammen.

Der „Static Model Engineer“ erweitert dann in der Aktivität „Design Business Objects“ das Business Object Modell um Attribute und Operationen von Business Entity und Business Service Objects, fügt bei Bedarf zusätzliche Business Service Objects hinzu und bestimmt die möglichen Eventkanäle zwischen den Business Objects. Dazu werden Business-Object-Diagramme des gesamten Business Object Modell verwendet.

Am Ende des Design-Workflows erstellt der „Infrastructure Manager“ auf Basis des neuen Business Object Modells in der Aktivität „Outline Deployment“

einerseits einen Deployment Plan für die hinzugekommenen Business Objects, andererseits einen Satz von Testfällen zur Sicherstellung von nicht-funktionalen Eigenschaften des Business Object Modells. Form und Darstellung des Deployment Plans und der Testfälle sind in diesem Vorgehensmodell nicht festgelegt.

8.3.4 Workflow: Implementation

Nachdem das Business Object Modell im Design-Workflow nahezu vollständig spezifiziert wurde, werden im Implementation-Workflow die noch zu einem vollständigen Modell fehlenden Methoden der Business Objects implementiert. In dieser Arbeit wird dazu ein Subset der Sprache Java verwendet.

Das vollständige Modell kann dann im Sinne des im nächsten Kapitel vorgestellten Werkzeugkonzepts auf eine Business Object Infrastruktur abgebildet und als lauffähiges Business Object System in Betrieb genommen werden.

8.3.4.1 Produkte

Übernommen aus vorherigen Workflows wird das Business Object Modell und der Deployment Plan. Aus vorigen Zyklen des Vorgehensmodells werden das aktuelle Infrastrukturmodell und das aktuelle Business Object System übernommen.

Erzeugt werden in diesem Workflow das fertige, erweiterte Business Object Modell, das durch die in Kapitel 7 vorgestellten Beschreibungstechniken dargestellt wird. Außerdem das neue, komplettierte Infrastrukturmodell und natürlich das fertige Business Object System.

Das Infrastrukturmodell wird im nächsten Kapitel genauer beschreiben und ist Kern des dreischichtigen Werkzeugkonzepts dieser Arbeit. Zweck des Infrastrukturmodells, das in XML-Form vorliegt, ist es, das Business Object Modell an die Business Object Infrastruktur anzupassen, ohne das Modell – ganz im Sinne der Kriterien aus Abschnitt 3.5 – von Infrastrukturelementen abhängig zu machen.

8.3.4.2 Rollen

An dem Implementation-Workflow sind drei neue Rollen beteiligt: „Component Engineer“, „Deployment Engineer“ und „Infrastructure Engineer“. Die Auftei-

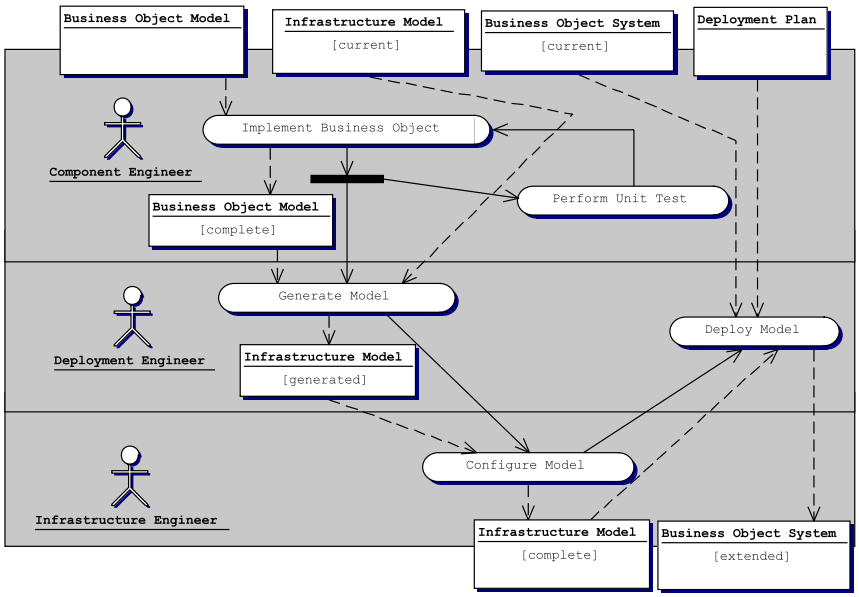

Abbildung 8.8: Implementation Workflow

lung in diese drei Rollen ergibt sich auch aus dem dreischichtigen Werkzeugkonzept.

Der „Component Engineer“ wurde aus dem generischen Vorgehensmodell des Unified Process übernommen. Seine Aufgabe ist es, die Methoden der Business Objects in der Zielsprache zu implementieren und für jedes einzelne Business Object sogenannte „Unit Tests“ auszuführen.

Der „Deployment Engineer“ ist Spezialist für den Übergang des Business Object Modells in ein Business Object System. Dazu verwendet er die im nächsten Kapitel vorgestellten Werkzeuge zur Verwaltung und Generierung des Infrastrukturmodells und des Werkzeugs zur Generierung des eigentlichen Business Object Systems.

Bevor das Infrastrukturmodell auf eine Business Object Infrastruktur deployed wird, konfiguriert ein „Infrastructure Engineer“ das Modell für eine bestimmte Infrastruktur beziehungsweise Middleware. Diese Rolle wird zusätzlich zur „Deployment Engineer“ Rolle eingeführt, weil es unter Umständen verschiedene Business Object Infrastrukturen gibt, für die jeweils Spezialisten gefragt sind. Für einen „Deployment Engineer“ ist die konkrete Infrastruktur jedoch egal.

8.3.4.3 Aktivitäten

Die fünf Aktivitäten „Implement Business Object“, „Perform Unit Test“, „Generate Model“, „Configure Model“ und „Deploy Model“ werden in Abbildung 8.8 dargestellt.

Der Inhalt der Aktivität „Implement Business Object“ wurde oben schon dargestellt. Die darauf folgende Aktivität „Perform Unit Test“ wurde vom generischen Unified Process übernommen. Im Gegensatz zum generischen Vorgehensmodell werden hier aber nicht die einzelnen Klassen beziehungsweise Komponenten, sondern die einzelnen Business Objects getestet. Dazu ist eine Simulationsumgebung für Business Objects notwendig, da das Business Objects Modell nur nach Abbildung auf die Infrastruktur ausführbar ist, was erst in den späteren Schritten des Workflows geschieht. Ein solcher Simulator ist nicht Teil des vorgestellten Frameworks und wird im Ausblick der Zusammenfassung als wünschenswerte Ergänzung des Frameworks vorgeschlagen.

Nachdem in sich geschlossene Teile des Business Object Modells fertig implementiert wurden, wird in der nächsten Aktivität „Generate Model“ vom „Deployment Engineer“ das Infrastrukturmodell generiert. Dazu wird als Ausgangspunkt das bestehende Infrastrukturmodell vorheriger Zyklen als Basis genommen und die in diesem Zyklus geänderten oder neuen Teile des Business Object Modells dazugefügt.

Diese Teile enthalten zunächst nur die fachlichen Informationen aus dem Geschäftsbereich und müssen im folgenden Schritt „Configure Model“ vom „Infrastructure Engineer“ um Elemente angereichert werden, die eine Abbildung der Business Objects auf die Ziel-Infrastruktur ermöglichen. Solche Elemente können zum Beispiel die Angabe von Datenbank-Tabellen oder die Angabe von Abbildungsregeln zur Aufteilung eines Business Objects auf mehrere Infrastrukturkomponenten sein.

Im abschließenden Schritt „Deploy Model“ wird das komplettierte Infrastrukturmodell vom „Deployment Engineer“ auf die Infrastruktur abgebildet und somit die neuen oder geänderten Komponenten für das bestehende Business Object System erzeugt. Das komplette Infrastrukturmodell und das erweiterte Business Object System sind dann Eingangsprodukte für den nächsten Zyklus.

8.3.5 Workflow: Test

In den Workflows Requirements und Design werden Testfälle für das System definiert, die im Test-Workflow ausgeführt werden. Dazu müssen zunächst spezielle Test-Clients entwickelt werden und dann anhand der Testfälle ausgeführt und evaluiert werden.

8.3.5.1 Produkte

Benötigt werden für diesen Workflow die Testfälle und das Business Object Modell und das zu testende Business Object System. Erzeugt werden „Test Components“, „Defects“ und eine „Test Evaluation“.

Test Components sind Clients des Business Object Systems und arbeiten auf den zu testenden Business Objects. Defects sind Fehlerprotokolle ohne festgelegte Darstellungsform. Die Test Evaluation ist das Ergebnis dieses Workflows und enthält eine Übersicht, welche Business Objects fehlerhaft sind und bei der nächsten Iteration korrigiert werden müssen.

8.3.5.2 Rollen

An dem Workflow sind beteiligt der „Test Engineer“, der „Component Engineer“ und der „System Tester“. „Test Engineer“ und „System Tester“ entsprechen den Rollen des Test-Workflows des generischen Unified Process. Auf die Rolle „Integration Tester“, wie sie im Test-Workflow des generischen Unified Process vorgeschlagen wurde, kann im Falle von Business Object Systemen verzichtet werden, da es keine Integration von separat entwickelten Komponenten gibt. Die Rolle „Component Engineer“ wurde bereits weiter oben vorgestellt.

Der „Test Engineer“ plant und entwirft anhand der Testfälle und des Business Object Modells die Test Components. Er hat dabei die Wahl zwischen verschiedenen Testmethoden, wie White-Box-, Black-Box- oder Pfad-Überdeckungstest. Die genaue Art der Tests ist je nach Testfall verschieden und kann im Vorgehensmodell nicht festgelegt werden. Eine gute Übersicht über die verschiedenen Testmöglichkeiten bietet [KFN99]. Er ist auch dafür verantwortlich, die Tests auszuwerten und eine Test Evaluation zu erstellen.

Der „System Tester“ führt die geplanten Tests aus. In den meisten Fällen werden „System Tester“ und „Component Engineer“ ein und dieselbe Person sein, da sich

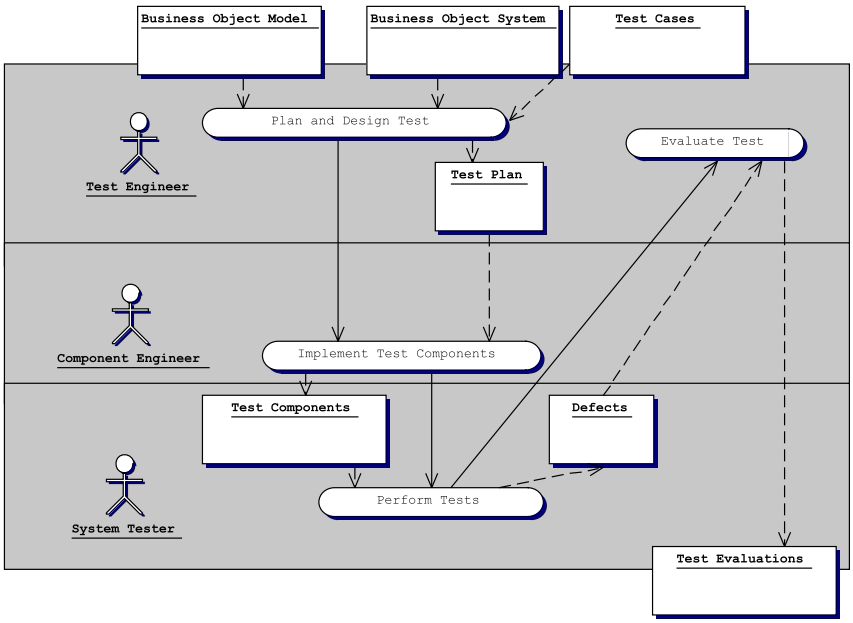

Abbildung 8.9: Test Workflow

der „Component Engineer“ am besten mit seinen Test Components auskennt, die – da sie nur in diesem Workflow verwendet werden – in der Regel keine ausgefeilte Benutzerschnittstelle besitzen.

8.3.5.3 Aktivitäten

Der Test-Workflow besteht aus vier Aktivitäten, die in Abbildung 8.9 dargestellt sind: „Plan und Design Test“, „Implement Test Components“, „Perform Tests“ und „Evaluate Test“.

Der Workflow beginnt mit der Aktivität „Plan and Design Test“, in der der Test Engineer auf Basis der Testfälle und des Business Object Systems die für die Tests notwendigen Testkomponenten entwirft.

Diese werden dann in der folgenden Aktivität „Implement Test Components“ vom Component Engineer implementiert und bilden die Grundlage für den nächsten Schritt „Perform Tests“. In dieser Aktivität führt der System Tester die Testfälle mithilfe der Test Components durch. Bei Fehlern werden Fehlerlisten („Defects“) generiert.

Im letzten Schritt des Workflows bewertet der Test Engineer die Testfälle anhand

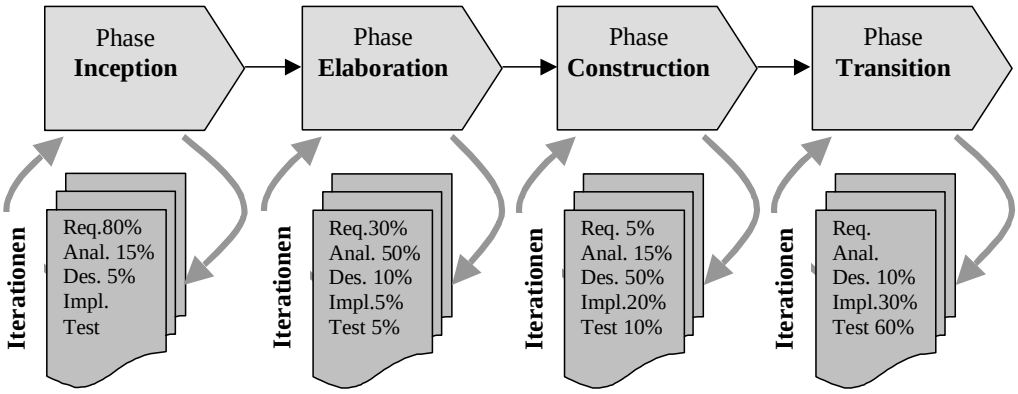

Abbildung 8.10: Phasen des Vorgehensmodells

der Fehlerlisten und erstellt ein Testgutachten mit den Anweisungen, welche Business Object fehlerhaft sind und in der nächsten Iteration korrigiert werden müssen.

8.3.6 Phasen

Die im vorigen Abschnitt vorgestellten Workflows sind die Kernelemente des Vorgehensmodells zur Entwicklung von unternehmensweiten Systemen auf Basis von Business Objects. Sie sind hauptsächlicher Bestandteil einer konkreten Instanz des Unified Process und damit eingebunden in dessen Prozeßframework. Das heißt, die Workflows sind nicht als eigenständige, sequentiell zu durchlaufende Abschnitte eines Softwareentwicklungsprozesses gedacht, sondern werden in jeweils mehreren Iterationen in den vier Phasen Inception, Elaboration, Construction und Transition mehrmals durchlaufen, wie es in Abbildung 8.10 gezeigt wird.

Dabei ist es aber nicht so, daß in jeder Iteration innerhalb einer Phase die Workflows komplett abgearbeitet und alle Ergebnisse des Workflows vollständig vorliegen. Stattdessen werden von Phase zu Phase die einzelnen Workflows unterschiedlich gewichtet und die Ergebnisse in den Iterationen inkrementell erarbeitet. In Abbildung 8.10 sind die Anteile der Workflows in den Phasen durch Prozentangaben gewichtet, wobei dies eine Empfehlung und keine Vorschrift ist.

Im folgenden werden die Aufgaben und die Ergebnisse der vier Phasen eines Vorgehensmodells zur Entwicklung von Business Object Systemen vorgestellt.

8.3.6.1 Phase: Inception

In der Phase Inception werden die Anforderungen an das neue beziehungsweise geänderte Business Object System erarbeitet. Dazu muß geklärt werden welche Teile des Geschäftsmodells überhaupt neu in das Business Object System aufgenommen oder geändert werden sollen. Diese Ergebnisse werden überwiegend im Requirements-Workflow erbracht.

Daneben wird der Analysis-Workflow und der Design-Workflow im geringen Umfang durchlaufen. Ziel ist es in dieser Phase zu klären, ob die geplanten Änderungen am System überhaupt machbar sind. So könnte eine Hinzunahme eines Business Objects die Änderung von sehr vielen weiteren, schon vorhandenen Business Objects bedeuten und damit in der dem Entwicklungszyklus vorgegebenen Zeit oder mit den zur Verfügung stehenden Bearbeitern nicht durchführbar sein.

Ein Konzept des Unified Process Frameworks ist der Meilenstein am Ende jeder Phase. Meilensteine definieren Ergebnisse, die zum Abschluß der Phase vorliegen müssen. Für die Inception-Phase für das in dieser Arbeit vorgestellte Vorgehensmodell sind dies:

- Eine Machbarkeitsstudie, die entweder das Vorhaben befürwortet, daß heißt das Projekt wird fortgesetzt, oder es wird neu aufgesetzt beziehungsweise abgebrochen.
- Ein vorläufiger Projektplan mit der Festlegung der beteiligten Personen und den Terminen für die Meilensteine der anderen Phasen.
- Die für eine Hinzunahme in das System oder zur Änderung identifizierten Business Objects als Teil des Geschäftsmodells. Dieses Ergebnis sollte alle beteiligten Business Workflow Objects, Business Actor Objects und Business Task Objects enthalten und in Form von Business-Use-Case-Diagrammen und Business-Assignment-Diagrammen vorliegen.

Dieser Meilenstein kann in mehreren Iterationen erreicht werden, in für jede Iteration bestimmte Zwischenergebnisse definiert werden, die sogenannten Increments. Ein Increment in dieser Phase könnten zunächst die Business-Use-Case-Diagramme und im nächsten Increment die dazugehörigen Business-Assignment-Diagramme sein.

8.3.6.2 Phase: Elaboration

In der Phase Elaboration werden die Anforderungen an das zu entwickelnde Business Object System weiter ausgearbeitet. Das heißt, es wird festgestellt, welche Funktionen und Eigenschaften die Business Objects des fertigen Systems aufweisen müssen. Außerdem wird die Basis für die Business Object Architektur gelegt, das heißt die grundlegenden Beziehungen und Abhängigkeiten der Business Objects untereinander. Dies geschieht vorwiegend im Analysis-Workflow, in dem die neue Struktur des Business Object Modells spezifiziert wird.

Außerdem wird der Requirements-Workflow weitergeführt, allerdings weniger umfangreich als in der Inception-Phase. Häufig ergibt die Analyse des Modells weitere Anforderungen an das Systems, die den Ergebnissen, die im Requirements-Workflow erarbeitet wurden, hinzugefügt werden müssen.

In dieser Phase werden bereits erste Business Objects prototypisch entworfen, implementiert und getestet. Dies dient vor allem der frühen Einbindung derjenigen Projekte, die die Änderung oder Ergänzung des Business Object Systems überhaupt angestoßen haben, typischerweise Projekte zur Erstellung von Business Object Anwendungen. Die prototypischen Business Objects können zusammen mit den Prototypen der Business Object Anwendungen getestet werden. Aus diesem Grund werden in der Elaboration-Phase auch der Design-, Implementation- und Test-Workflow durchlaufen.

Für den Meilenstein zur Abschluß der Elaboration-Phase werden die folgenden Ergebnisse erwartet:

- Ein vervollständigter Projektplan mit der Aufteilung und zeitlichen Einteilung der Entwicklung der zu erstellenden oder zu ändernden Business Objects auf die Mitglieder des Projektteams.
- Beschreibung der neuen Business Object Architektur mit den neuen Business Objects und deren Beziehungen. Also alle Business Entity Objects, Business Service Objects und deren Generalisierungen und Assoziationen. Diese Beschreibung liegt in Form von Business-Object-Diagrammen vor.
- Die fachlichen Testfälle für die neuen oder geänderten Geschäftsvorfälle.

Diese Ergebnisse können inkrementell so erreicht werden, indem die Business Object Architektur für jeden Geschäftsvorfall separat erweitert wird. Es werden also die neuen Geschäftsvorfälle nacheinander analysiert.

8.3.6.3 Phase: Construction

Das Ziel der Construction-Phase ist ein wichtiger Meilenstein im Entwicklungsprozeß: Das lauffähige, geänderte oder erweiterte Business Object System, das sogenannte Beta-Release. Dazu werden hauptsächlich im Design- und Implementation-Workflow die Business Objects ausgearbeitet, das heißt, die Methoden für die Operationen implementiert und die Abläufe der Business Object Workflows festlegt. Weiterhin werden die fachlichen Testfälle aus der Phase Inception ergänzt um nicht-funktionale Testfälle zur Sicherstellung von Quality-of-Service-Eigenschaften des Systems. Schließlich wird das komplette Business Object Modell auf eine Infrastruktur abbildet um das Beta-Release des Systems zu erzeugen.

Auch die Aktivitäten im Test-Workflow werden verstärkt durchgeführt. In dieser Phase können bereits einige Testfälle durchgeführt werden, die sich auf abgeschlossene, bereits fertiggestellte Teile des Business Object Systems beziehen.

Neben dem erwähnten Beta-Release, sind folgende Ergebnisse Bestandteil des Meilensteins der Construction-Phase:

- Das vollständige Business Object Modell aus den fünf in Kapitel 7 vorgestellten Sichten. Das bedeutet, daß das Business Object Modell in wechselseitig konsistenten Business-{Use-Case- | Assignment- | Object- | Workflow- | Organization}-Diagrammen vorliegen muß.
- Ein kompletter Satz fachlicher und Quality-of-Service-Testfälle.
- Das um die Infrastrukturaspekte erweiterte Infrastrukturmodell für das Business Object System in der Beta-Release.

Die Construction-Phase kann in einzelne Iterationen zerlegt werden, indem man nacheinander nicht oder nur wenig zusammenhängende Business Objects Strukturen implementiert.

8.3.6.4 Phase: Transition

Die Transition-Phase ist geprägt vom Übergang des Business Object Systems vom Beta-Release bis zur endgültigen Freigabe und Inbetriebnahme beziehungsweise Ersatz des bestehenden Systems. Hauptsächlich werden im Test-Workflow die in den früheren Phasen spezifizierten Testfälle abgearbeitet und – falls Fehler

vorliegen – die entsprechenden Design- und Implementation-Workflows für die fehlerhaften Business Objects angestoßen.

Diese Phase ist bei Business Object Systemen besonders wichtig, da das System bei einer vollständigen Umsetzung des Business Object Paradigma für die Funktion aller IT-unterstützen Geschäftsvorfälle verantwortlich ist. Fehler im Business Object System wirken sich unter Umständen auf den Betrieb des ganzen Unternehmens aus. Deshalb werden in der Transition-Phase nicht nur die Fehler der neuen oder geänderten Business Objects beseitigt, sondern mit umfangreichen Regressionstest sichergestellt, daß alle bisherigen Business Object Anwendungen noch einwandfrei funktionieren.

In dieser Phase müssen auch die Projekte eingebunden werden, die die Erweiterung oder Änderung des Business Object Systems angestoßen haben. Dazu werden die Beta-Releases dieser Projekte zusammen mit dem Business Object System getestet. Die Testfälle richten sich nach den in den Projekten definierten Testfälle.

Meilenstein für die Transitionsphase ist die Inbetriebnahme des Business Object Systems oder der Ersatz eines bestehenden Business Object Systems. Neben dem System gehört als Ergebnis des Meilensteins das vollständige Business Object Modell und das optimierte Infrastrukturmodell.

Aufgeteilt werden kann diese Phase in Iterationen, die zunächst die fachlichen Testfälle, danach die Quality-of-Service-Testfälle und schließlich die Regressionstests durchführen. Denn es macht wenig Sinn ein System zu optimieren, das noch fachliche Fehler aufweist.

8.4 Business Object Anwendungsentwicklung

Die Entwicklungen von Business Object Anwendungen, also Systemen, die auf dem Business Object Modell arbeiten, ist nicht unabhängig von dem im vorigen Abschnitt vorgestellten Vorgehen. Im Gegenteil: der Bedarf nach einer neuen Business Object Anwendung stößt in der Regel die Änderung oder Ergänzung des Business Object Systems – also den Software-Entwicklungszyklus der oben vorgestellt wurde – an und ist in bestimmten Phasen, Workflows und Aktivitäten mit ihm verzahnt.

In Abbildung 8.11 ist ein Beispiel eines Business Object Systems mit drei Anwendungen dargestellt. Wenn zum Beispiel Anwendung C neu entwickelt wird,

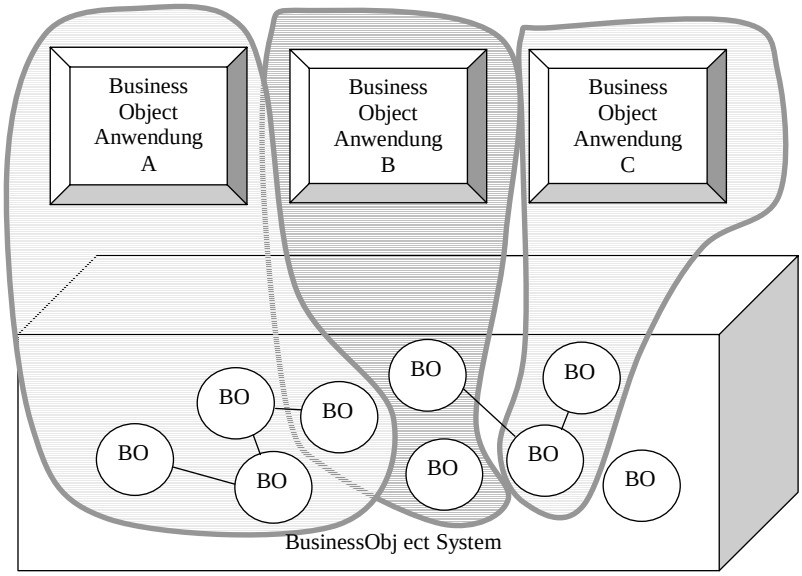

Abbildung 8.11: Abhängigkeiten zwischen BO-System und -Anwendungen

müssen zwei Business Objects und eine Assoziation zu einem bereits bestehenden Business Object in das Business Object System hinzugefügt werden. Würde Anwendung B neu erstellt, müßten nicht nur zwei neue Business Objects hinzugefügt werden, sondern unter Umständen ein bereits bestehendes Business Object, das mit der Anwendung A geteilt wird, geändert werden.

Da Business Object Anwendungen bis auf die Anbindung an das Business Object System traditionelle Client/Server Systemen entsprechen, können sie größtenteils im Rahmen eines traditionellen Vorgehens wie Catalysis [DW99] oder dem Rational Unified Process [Kru98] entwickelt werden.

In diesem Abschnitt werden die Anknüpfungspunkte und Änderungen eines solchen Vorgehensmodelles an den Entwicklungsprozeß des Business Object Systems und die notwendigen Ergänzungen um neue Aktivitäten und Zuständigkeiten dargestellt.

Da die Verzahnungen der Entwicklung von Business Object Systemen und Business Anwendungssystemen je nach gewählten Vorgehensmodell unterschiedlich sein können, wird in dieser Arbeit eine Erweiterung des Rational Unified Process zur Entwicklung von Business Anwendungssystemen diskutiert. Die dargestellten Ergänzungen sind aber auch auf andere Vorgehensmodelle anwendbar.

Der Rational Unified Process (RUP) ist wie Anfang des Kapitels bereits erwähnt eine Instanz des Unified Process und dient als Vorgehensmodell zur Entwick-

lung von komponentenbasierten Client-/Server-Informationssystemen auf Basis der UML und objektorientierter Programmiersprachen. Eine genauere Beschreibung des RUP findet sich in [Kru98].

Da der RUP wie das Vorgehen zur Entwicklung von Business Object Systemen auf dem Unified Process basiert, sind große Teile des Prozeßframeworks ähnlich. Der RUP ist ebenfalls in vier Phasen Inception, Elaboration, Construction und Transition und den Iterationen innerhalb der Phasen eingeteilt. Ferner sind Workflows, Aktivitäten und Produkte definiert, die sich auf Grund der verschiedenen Entwicklungsszenarien unterscheiden.

8.4.1 Workflows

Im folgenden werden die Workflows des RUPs mit dem Ziel betrachtet, Anknüpfungspunkte zum Vorgehensmodell zur Business Object Systementwicklung aufzuzeigen. Für eine komplette, umfassende Darstellung der Workflows wird auf [Kru98] verwiesen.

8.4.1.1 Workflow: Business Modeling

Am Anfang der Workflows im RUP steht die Modellierung des Geschäftsbereiches. Ziel ist die Modellierung des Business Use Case- und des Business Object-Modells des Anwendungsbereichs. Dieser Workflow ist in einem Unternehmen mit einem unternehmensweiten Ansatzes auf Basis von Business Objects nicht notwendig. Die genannten Modelle stehen bereits als Ergebnis eines Business Process Reengineering zur Verfügung oder liegen bereits als Business Object Modell vor, da bereits Bestandteil des Business Object Systems.

Dieser Workflow sollte ersetzt werden durch die Analyse und Identifikation der benötigten Elemente des Business Process Reengineering- oder Business Object Modells. Daraufhin sollte falls nötig (neue Business Objects werden benötigt oder vorhandene müssen angepaßt werden) die Änderung oder Ergänzung des Business Object Systems getriggert werden. Die Ergebnisse der Analyse und Identifikation sind also Eingangsprodukte und Startpunkt für den Workflow Requirements der Business Object System-Entwicklung. Dieser Entwicklungsprozeß läuft dann parallel zum RUP.

8.4.1.2 Workflow: Requirements

Der Workflow Requirements im RUP dient im wesentlichen zur Modellierung der Nutzungsfälle des Systems. Dazu werden die späteren Nutzer des Systems befragt unter Zuhilfenahme von Benutzeroberflächen-Prototypen. Außerdem wird in diesem Workflow der Projektplan mit den Schätzungen für die benötigte Zeit und die benötigten Bearbeiter festgelegt.

Diese Schätzungen sind auch der einzigste Anknüpfungspunkt zum parallel laufenden Entwicklungsprozeß zur Änderung oder Erweiterung des Business Objects Systems. Die dort in der Inception-Phase ermittelten Termine für den Beta-Release beziehungsweise zur Fertigstellung des Business Objects Systems beeinflussen direkt den Projektplan, denn ohne fertiges Business Object System kann die Business Object Anwendung nicht umfassend getestet oder in Betrieb genommen werden.

8.4.1.3 Workflow: Analysis/Design

Analyse und Design sind im RUP zusammengefaßt, da davon ausgegangen wird, das es – gerade in objektorientierten Systemen – keine logische Trennung zwischen Analyse- und Designmodell gibt, sondern der Übergang fließend ist. Design wird als Verfeinerung der Analyse aufgefaßt.

Ergebnis des Workflows ist die Architektur des Systems, das Klassenmodell und das Design der benötigten Datenbank-Instanzen. Letzteres ist im Fall der Business Object Anwendungen nicht notwendig. Stattdessen sollte die Rolle „Database Designer“ ersetzt werden durch „Business Object Usage Designer“. Diese Rolle übernimmt das Design der Anbindung der Anwendung an das Business Object System. Dieser sollte ebenfalls ein Subsystem designen, das die Anbindung zum Business Object System simuliert, damit bestimmte Teile der Anwendung, zum Beispiel die Benutzerinteraktion, schon vor Fertigstellung des Business Object Systems im parallel laufenden Prozeß getestet werden können.

8.4.1.4 Workflow: Implementation

Im Implementation-Workflow werden die Klassen des Anwendungsmodells auf Komponenten und Subsysteme verteilt und implementiert. Diese Komponenten und Subsysteme werden zunächst einzeln getestet bevor sie anschließend zum kompletten Anwendungssystem zusammengesetzt werden.

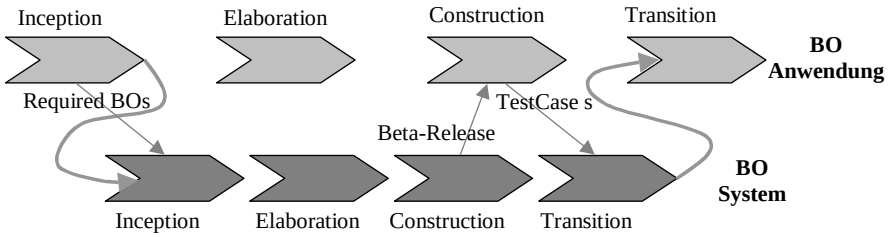

Abbildung 8.12: Verzahnung der Entwicklungsprozesse von Anwendung und System

Die Anbindung an die Business Object System-Entwicklung sind in diesem Workflow minimal. Solange das Business Object System noch nicht fertig ist, wird das Subsystem zur Anbindung der Anwendung an das Business Object System implementiert und integriert. Ist das Business Object System fertig, dann wird dieses Subsystem durch die eigentliche Anbindung ersetzt.

8.4.1.5 Workflow: Test

Der Test-Workflow dient im RUP zum Testen verschiedener funktionaler und nicht-funktionaler Eigenschaften der Anwendung. Trennen muß man hier zwischen den Eigenschaften, die keine echte Anbindung an das Business Object System zur Voraussetzung haben, wie einen Test der Benutzeroberfläche oder des Sessionmanagements und den Eigenschaften, die eine echte Anbindung an das Business Object System erfordern, zum Beispiel das Anstoßen und Ausführen von Workflows, Ausführen von komplexen Operationen auf den Business Objects oder Test von Zugriffs- und Ausführungszeiten.

Im letzten Fall muß das Testen der Anwendung solange zurückgestellt werden, bis das Business Object System zumindest als Beta-Release vorliegt. Dann allerdings sind die Ergebnisse dieser Test wiederum Eingangsdaten für die Transition-Phase des Business Object Systems.

8.4.2 Phasen

Die Verknüpfung der Phasen beider Entwicklungsprozesse ergibt sich aus den Zusammenhängen der Workflows. Abbildung 8.11 zeigt das Zusammenspiel jeweils eines Entwicklungszyklus zur Entwicklung einer Business Object Anwen-

dung und die von diesem angestoßene Änderung beziehungsweise Ergänzung des Business Object Systems.

Nach der Inception-Phase der Anwendungserstellung wird beginnt die (Weiter-) Entwicklung des Business Object Systems mit dessen Inception-Phase. Eingangsdaten für die Entwicklung des Business Object Systems ist die Anforderung mit den benötigten Änderungen an bestehenden Business Objects oder den benötigten neuen Business Objects.

Das Beta-Release des Business Object Systems ist notwendig für bestimmte Testfälle des Anwendungssystems. Fehler und Unstimmigkeiten, die bei diesen Testfällen auftreten, werden – soweit vom Business Object System verursacht – in der Transition-Phase des Business Object Systems behoben. Sobald das Business Object System fertiggestellt wurde, kann auch das Anwendungssystem nach der Transition-Phase an die anfordernde Fachabteilung ausgeliefert werden.

8.5 Zusammenfassung

Die Entwicklung von Business Object Systemen und deren Anwendungen entspricht nicht dem klassischen Vorgehen zur Entwicklung von Informationssystemen. Wie die vom Business Object System abgebildete Geschäftswelt, befindet sich das System in einer kontinuierlichen, inkrementellen Entwicklung. Die derzeitigen Vorgehensmodelle können ohne eine Anpassung an die Besonderheiten des Business Object Paradigma nicht angewendet werden.

In diesem Kapitel wurden die Besonderheiten einer Entwicklung von unternehmensweiten Anwendungen auf Basis von Business Objects herausgearbeitet. Unter Berücksichtigung dieser Besonderheiten wurde ein Vorgehensmodell auf Basis des Unified Process vorgestellt. Es erweitert den Unified Process um diejenigen Workflows, Produkte und Akteure, die notwendig sind, ein Business Object System auf Basis des in dieser Arbeit vorgestellten Frameworks, das heißt dem Metamodell, den Beschreibungstechniken und dem Werkzeugkonzept, zu entwickeln.

Außerdem wurde gezeigt, wie man den Rational Unified Process nutzen kann, um Business Object Anwendungen zu entwickeln. Beide Entwicklungsprozesse – die des Business Object Systems und der Business Object Anwendung – sind miteinander verzahnt. Die entsprechenden Anknüpfungspunkte des RUP an ein unternehmensweites Vorgehen zur Entwicklung von Business Objects wurden offengelegt.

Das in diesem Kapitel entwickelte Vorgehensmodell ist vor allem darauf ausgelegt, daß dem Entwickler geeignete Werkzeuge zur Modellierung des Business Object Modells und zur Generierung des Business Object Systems zur Verfügung stehen. Im nächsten Kapitel wird ein Konzept für diese Werkzeuge vorgestellt.

Kapitel 9

Ein integriertes Werkzeugkonzept

In diesem Kapitel wird das methodische Framework durch ein integriertes Werkzeugkonzept für die Entwicklung offener, unternehmensweiter Systeme erweitert. Die Ergebnisse dieses Kapitels umfassen:

- Eine dreistufige Architektur für eine Werkzeugumgebung zur Entwicklung von Business Object Systemen.
- Eine Konzeption eines Modellierungswerkzeugs zur Darstellung und Bearbeitung der verschiedenen Sichten auf das Business Object Modell.
- Die eXtended Business Object Definition Language (XBODL) zur Definition eines werkzeugunabhängiges Infrastrukturmodells.
- Ein Vorschlag zur Abbildung des Business Object Modells auf die Enterprise JavaBeans Infrastruktur.
- Eine Realisierung des Werkzeugkonzepts am Beispiel einer prototypischen Werkzeugumgebung zur Entwicklung von Business Object Systemen.

9.1 Anforderungen

Aufgrund der zunehmend modellbasierten Entwicklung von Softwaresystemen werden Softwareentwicklungswerkzeuge verstärkt eingesetzt. Wurden bisher Werkzeuge vor allem in den späteren Phasen, vor allem während der Implementierung, genutzt, so werden nun auch in den früheren Phasen Analyse und Design

vor allem Modellierungswerkzeuge für graphische Beschreibungstechniken eingesetzt.

In der Regel führt dies zu einer heterogenen und wenig integrierten Werkzeugumgebung, bestehend aus Werkzeugen zur Geschäftsprozeß-, zur Workflow- und zur objektorientierten Modellierung, sowie aus Programmiersprachen-, Datenbankdefinitions- und Codegenerierungs-Werkzeugen.

Das grundlegende Element des in dieser Arbeit vorgestellten Frameworks ist das Business Object Metamodell. Es ist Basis für die Beschreibungstechniken, das Vorgehensmodell und die Business Object Systeme selbst. Für die Modellierung des Systems definiert es die erlaubten Elemente, ihre gegenseitigen Abhängigkeiten und Navigations- und Abbildungsmöglichkeiten. Für das Business Object System definiert es die Aufteilung in verschiedene Komponenten. Ohne eine integrierte Werkzeugunterstützung zur Sicherstellung eines korrekten und konsistenten Business Object Modells und seiner automatischen Generierung und Abbildung auf eine Infrastruktur wäre ein unternehmensweites Vorgehen auf Basis von Business Objects nur schwer durchführbar.

Deshalb wird das Framework abschließend ergänzt durch ein integriertes Werkzeugkonzept. Es umfaßt die Konzeption eines UML-basierten Modellierungswerkzeugs auf Basis eines kommerziellen UML-Tools zur Darstellung und Bearbeitung der verschiedenen Sichten auf das Business Object Modells. Weiterhin wird ein werkzeugunabhängiges Infrastrukturmodell als Zwischendarstellung und Weiterbearbeitung des Business Object Modells zur deklarativen Erweiterung des Modells um technische Aspekte vorgestellt. Das Konzept wird ergänzt durch einen Vorschlag zur Abbildung des Infrastrukturmodells auf eine geeignete Infrastruktur – in der Arbeit wurde dazu das Enterprise JavaBeans-Framework gewählt.

Am Schluß des Kapitels wird eine Realisierung des Konzepts anhand einer in einer Diplomarbeit [Frö01] entstandenen prototypischen Werkzeugumgebung zur Entwicklung von Business Object Systemen vorgestellt.

9.2 Architektur

Das Werkzeugkonzept basiert auf einer dreischichtigen Architektur der Werkzeugumgebung, wie sie in Abbildung 9.1 dargestellt ist: In der obersten Schicht wird das Business Object Modell mithilfe von an das Metamodell angepaßten, graphischen Modellierungswerkzeugen in Form von Business-Use-Case-, Business-

Assignment-, Business-Object-, Business-Workflow- und Business-Organization-Diagrammen bearbeitet. Das Werkzeug erlaubt die Darstellung des Business Object Modells in den fünf Sichten und stellt gleichzeitig die Konsistenz des Modells her, das in den unterschiedlichen Diagrammen spezifiziert wird. Außerdem erlaubt es, zwischen den Diagrammen anhand gleicher Metamodellelemente zu navigieren. Das Konzept für ein solches Werkzeug wird in Abschnitt 9.3 detailliert vorgestellt.

Die Abbildung des Business Object Modells dieser Schicht auf das Infrastrukturmodell der mittleren Schicht leistet ein Generator. Er setzt die (konsistenten) Elemente der Diagramme um in eine Zwischendarstellung.

Diese Zwischendarstellung, das Infrastrukturmodell, ist der Integrationspunkt der mittleren Schicht. Das Infrastrukturmodell ist eine Instanz der XML und dient der Erweiterung des Business Object Modells um technische oder infrastrukturelle Eigenschaften. Es kann mit beliebigen Werkzeugen zur Bearbeitung von XML-konformen Modellen geändert werden. Wie das Infrastrukturmodell im Detail aufgebaut ist, wird in Abschnitt 9.4 beschrieben.

Auch zwischen der mittleren Schicht und der untersten Schicht mit den Laufzeitkomponenten des eigentlichen Business Object Systems wird das Infrastrukturmodell durch einen Generator auf das Infrastrukturframework abgebildet. Wie eine solche Abbildung im Falle des Enterprise JavaBeans-Frameworks aussieht, wird im Abschnitt 9.5 beschreiben.

9.3 Modellierungswerkzeug

In diesem Abschnitt wird ein Konzept für ein UML-basiertes Modellierungswerkzeug zur Darstellung und Bearbeitung eines Business Object Modells vorgestellt. Ausgangspunkt des Konzepts ist die Annahme, daß ein völlig neu zu entwickelndes Werkzeug für die Business Object Modellierung nicht notwendig ist. Der Grund dafür ist die Beschränkung des in Kapitel 7 vorgestellten Profils der UML zur Modellierung von Business Objects auf die im UML-Standard vorgesehenen Erweiterungstechniken und Diagrammarten. Dadurch ist es möglich, die meisten, kommerziellen UML-Werkzeuge, die diese Erweiterungsmechanismen unterstützen, anzupassen.

Diese Anpassung muß neben der Darstellungsmöglichkeit für die fünf in Kapitel 7 definierten Diagrammarten auch eine Möglichkeit zur Konsistenzprüfung des gesamten Business Object Modells geben. Zudem ist für einen praktikablen

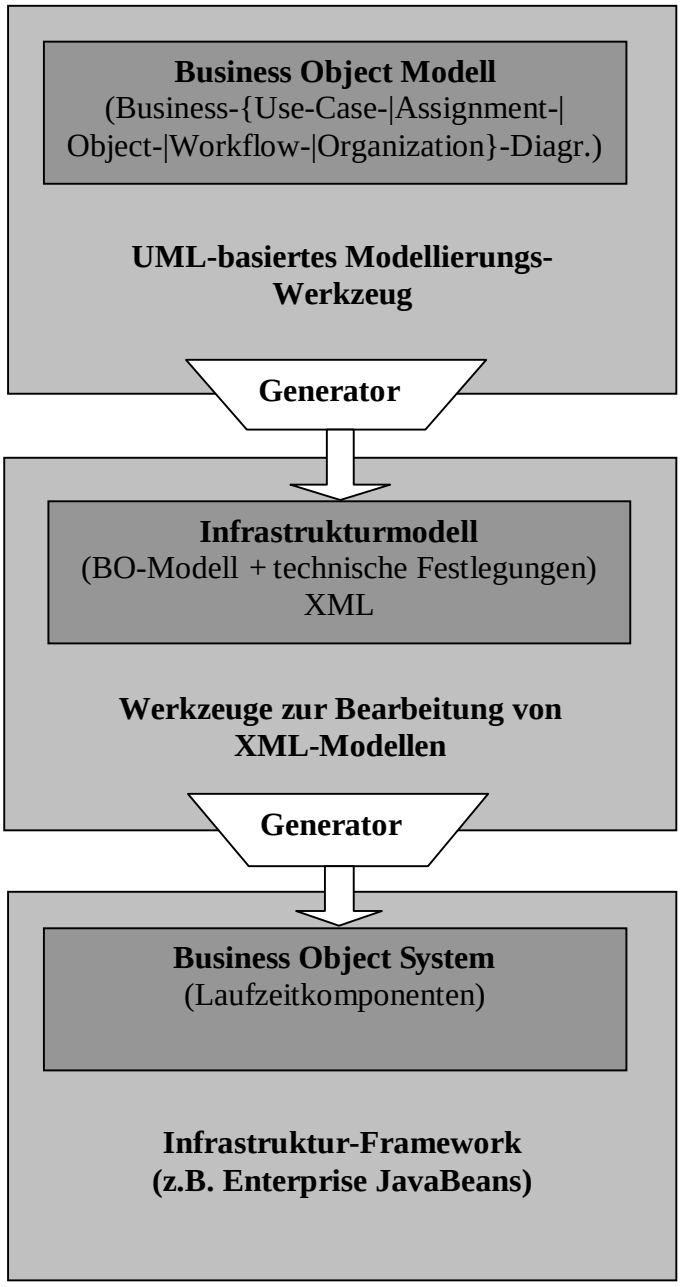

Abbildung 9.1: Integriertes Werkzeugkonzept

Umgang mit einem umfangreichen Modell darauf zu achten, daß dem Entwickler Navigationsmöglichkeiten zwischen den Diagrammen zur Verfügung stehen.

Diese Anforderungen stehen den Möglichkeiten zur Erweiterung der verschiedenen, auf dem Markt erhältlichen Werkzeuge gegenüber. Im nächsten Abschnitt werden exemplarisch die Erweiterungsmöglichkeiten des UML-Werkzeugs Together und eine mögliche Umsetzung der Anforderungen gezeigt.

9.3.1 Erweiterungsmöglichkeiten

Auf dem Markt befinden sich derzeit drei UML-Werkzeuge mit einem großen Anwenderkreis: Together von der Firma Togethersoft, Rose von der Firma Rational und Software-trough-Pictures der Firma Aonix.

Alle drei bieten Schnittstellen – sogenannte Application Programming Interfaces (APIs) – zur Anpassung und Erweiterung ihrer Funktionalität. Alle drei erlauben es, UML-Diagramme mithilfe von Stereotypen, Tags und Constraints an ein UML-Profil anzupassen.

Im Rahmen dieser Arbeit wird das API des Werkzeugs Together genauer vorgestellt und gezeigt, wo Möglichkeiten für eine Umsetzung der Anforderungen an ein Werkzeug zur Modellierung eines Business Object Modells vorhanden sind. Together wurde deshalb ausgewählt, weil es als einzigstes der drei Tools eine API für eine verbreitete, standardisierte Programmiersprache, nämlich Java, bietet. Die Erweiterungsschnittstellen der beiden anderen Werkzeuge basieren dagegen auf proprietären Skriptsprachen.

9.3.1.1 Die Erweiterungsschnittstelle von Together

Together bietet dem Entwickler umfangreiche und komfortable Möglichkeiten, das Werkzeug zu erweitern. Dazu werden drei grundsätzliche APIs zur Verfügung gestellt:

- Das Integrated Development Environment (IDE) API bietet die Möglichkeit, die Präsentation der Modellelemente sowie die Benutzeroberfläche von Together zu beeinflussen.
- Das Source Code Interface (SCI) API gestattet den Zugriff auf den von Together für verschiedene UML-Elemente (Klassen, Assoziationen, Attribute,

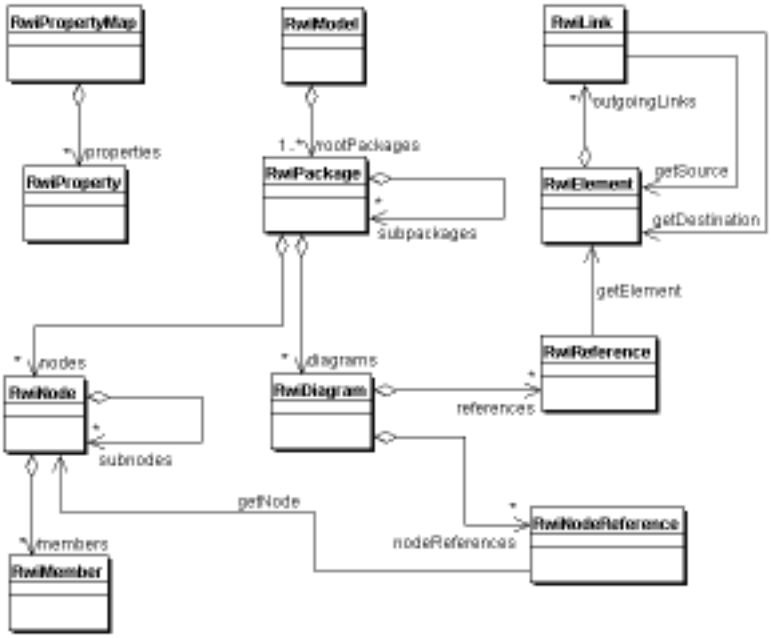

Abbildung 9.2: Das Together Read-Write Interface (aus [Tog01])

Methoden) automatisch generierten Quelltext. Dies ist eine exklusive Eigenschaft von Together und soll das Round-Trip-Engineering von Modell und Quelltext einfacher gestalten.

- Das Read-Write Interface (RWI) API ermöglicht den Zugriff auf alle Modellelemente, die durch die vom Benutzer erstellten Diagramme dargestellt werden.

Das letzte API ist ein geeigneter Anknüpfungspunkt zur Erweiterung der Together-Funktionalität zur Manipulation, Generierung und Konsistenzprüfung von Business Object Modellen.

Der generelle Aufbau des RWI-APIs ist in Abbildung 9.2 dargestellt. Wichtig für eine Erweiterung ist vor allem das Element `RwiNode`. Das Element `RwiNode` steht für alle im Modell enthaltenen Elemente. Die Unterscheidung, um welches Element (Klasse, Assoziation, Use Case usw.) es handelt, wird über Eigenschaft-Wert-Paare (Properties) festgelegt.

Die Vorgehensweise zur Implementierung beispielsweise von Konsistenzprüfungen wäre folgende: Man durchläuft den Modellbaum, der durch `RwiModel` aufgespannt wird, `RwiNode` für `RwiNode`, und führt je nachdem, um welches Ele-

ment es sich handelt, die entsprechenden Prüfungen durch.

Im folgenden Quelltext-Fragment wird dies exemplarisch für ein Business Workflow Object, das im UML-Profil für das Business Object Modell ja einer Klasse mit dem Stereotyp „Business Workflow“ entspricht, dargestellt.

```
String shapetype
  = node.getProperty(RwiProperty.SHAPE_TYPE);
String stereotype
  = node.hasProperty(TaggedValue.STEREOTYPE) ?
    node.getProperty(TaggedValue.STEREOTYPE) : "";

if (shapetype.equals(RwiShapeType.CLASS)
    && stereotype.equals("Business Workflow"))
{
   // Überprüfe Business Object Workflow "node"
   // ...
}
```

Der Typ des Modellelements kann aus dem Property „SHAPE_TYPE“ ausgelesen werden. In diesem Fall wird nach einer Klasse, also dem Wert „RwiShapeType.CLASS“, gesucht. Zusätzlich werden Business Workflow Objects durch ein Stereotyp gekennzeichnet. Dies kann aus dem Wert des Property „TaggedValue.STEREOTYPE“ ermittelt werden. In diesem Fall eben „Business Workflow“. Ist das Element auf diese Weise identifiziert, können die Überprüfungen oder sonstigen Manipulationen durchgeführt werden.

Natürlich können auf diese Weise auch die anderen Erweiterungen des UML-Profils, die Tagged Values und die Constraints, abgefragt und entsprechend behandelt werden. Im Unterschied zu oben gezeigten Vorgehen bei einem Stereotyp ändert sich lediglich der gesuchte Eigenschaftstyp („TaggedValue.TAG“ und „TaggedValue.CONSTRAINT“).

Mithilfe des dargestellten RWI-APIs lassen sich damit alle Eigenschaften und Elemente des UML-Profils für das Business Object Modell im Werkzeug Together prüfen beziehungsweise bearbeiten. Eine prototypische Umsetzung der Konsistenzprüfungen und eines Generierungsmoduls für das Infrastrukturmodell auf Basis der hier vorgestellten Techniken wird in Abschnitt 9.6 vorgestellt.

9.3.1.2 Fehlende Eigenschaften

Obwohl die Together-API nahezu jegliche Manipulation eines UML-Modells zulassen, fehlen Eigenschaften, die für die Umsetzung der Anforderungen wünschenswert wären.

Zum Einen fehlt Together die Möglichkeit, das Modell mithilfe eines MOF-konformen APIs zu bearbeiten. Ein MOF-API, wie es die OMG vorschlägt [OMG99b], hätte den Vorteil, Konsistenz- und Generierungsmodule zu entwickeln, ohne von einem proprietären API wie dem Read-Write Interface abhängig zu sein. Die einmal entwickelten Module könnten dann ohne großen Aufwand an andere UML-Werkzeuge, die ein MOF-API anbieten, angepaßt werden. Das Fehlen einer solchen Möglichkeit ist in der derzeitigen Praxis jedoch wenig relevant, denn es existieren derzeit sowieso keine kommerziellen UML-Werkzeuge mit MOF-konformen API.

Zum Anderen wäre es sehr komfortabel, wenn Together einen integrierten OCL-Interpreter hätte. Dann könnten die OCL-Regeln für das in dieser Arbeit entwickelte Metamodell sofort übernommen werden. So muß der Entwickler die in Kapitel 6 definierten Regeln zunächst in Java unter Nutzung des RWI-APIs übersetzen. Es gibt zur Zeit jedoch nur einen Forschungsprototypen ARGO UML [RR00], der einen OCL-Interpreter für Konsistenzprüfungen auf dem Modell zur Verfügung stellt.

9.3.2 Konsistenzbedingungen

Für das Business Object Modell wurden in Kapitel 6 Regeln definiert. Diese Regeln stellen korrekte Instanzen des Metamodells sicher. Das heißt, nur wenn alle Bedingungen die in diesen Regeln definiert werden erfüllt sind, handelt es sich um ein gültiges Business Object Modell.

Diese Regeln gelten aber unmittelbar nur für die Metamodell-Elemente. Sie berücksichtigen nicht, daß das Modell durch fünf verschiedene Diagrammarten beschreiben werden kann. Im Metamodell gibt es keinen Begriff Diagramm (im Gegensatz zum UML-Metamodell).

Damit die Modellelemente, die in den Diagrammen modelliert werden, auch ein gültiges Business Object Modell darstellen, müssen zusätzliche, auf Diagramme bezogene Konsistenzregeln angegeben werden. Die Überprüfung dieser Regeln ist Aufgabe des Modellierungswerkzeugs.

Diagrammtyp		Konsistenzbedingung
Business-Use-Case-Diagr.	[1]	Alle Use-Case-Elemente sind mit dem Stereotyp „Business Workflow“ gekennzeichnet.
	[2]	Alle Actor-Elemente sind mit dem Stereotyp „Business Actor“ gekennzeichnet.
	[3]	Von einem Actor-Element existiert immer eine Assoziation zu einem Use-Case-Element.
Business-Assignm.-Diagr.	[4]	Alle Use-Case-Elemente sind mit dem Stereotyp „Business Task“ gekennzeichnet.
	[5]	Alle Actor-Elemente sind mit dem Stereotyp „Business Actor“ gekennzeichnet.
	[6]	Für jedes BAO gibt es ein entsprechendes BAO in einem Business-Use-Case-Diagramm.
Business-Object-Diagr.	[7]	Klassen können mit den Stereotypen „Business Actor“, „Business Task“, „Business Workflow“, „Business Service“, „Proxy“ oder ohne gekennzeichnet sein.
	[8]	Zu jedem BAO gibt es ein BAO in einem Business-Use-Case-Diagramm.
	[9]	Zu jedem BWO gibt es ein BWO in einem Business-Use-Case-Diagramm.
	[10]	Alle Klassen sind Bestandteil eines Packages (Domain).
Business-Workflow-Diagr.	[11]	Zu jedem Action-Element gibt es ein BTO im Business-Object-Diagramm.
	[12]	Zu jedem Activity-Element gibt es ein BWO im Business-Object-Diagramm.
	[13]	Ein ObjectFlow-Element kann optional mit dem Stereotyp „Business Actor“ gekennzeichnet sein.
	[14]	Eine Relationship zwischen einem ObjectFlow-Element und einem Action- bzw. Activity-Element ist nur in Richtung Action/Activity navigierbar.
Business-Organ.-Diagr.	[15]	Klassen können optional mit dem Stereotyp „Business Actor“ gekennzeichnet sein.
	[16]	Für Klassen, die mit dem Stereotyp „Business Actor“ gekennzeichnet sind, ist das Constraint „Role“ oder „Actor“ vorhanden.

Tabelle 9.1: Konsistenzbedingungen auf den Diagrammtypen

In Tabelle 9.1 werden die zusätzlichen Konsistenzregeln für die Diagrammdarstellung des Business Object Modells aufgelistet. Für alle Diagrammarten gilt ferner folgende Regel [0]: „Alle Namensbezeichner für die Modellelemente dürfen nur aus den Zeichen A-z, 0-9 und dem Unterstrich bestehen. Beginnen müssen die Namen mit einem Buchstaben. Enthaltende Leerzeichen werden als Unterstriche behandelt“.

9.3.3 Navigationsmöglichkeiten

Neben den Konsistenzbedingungen sollte das Modellierungswerkzeug auch Navigationsmöglichkeiten zwischen den Diagrammen bereitstellen. Together erlaubt eine Navigation durch die Angabe von sogenannten Hyperlinks. Mit Hyperlinks kann jedes Diagrammelement mit beliebig vielen anderen Elementen aus anderen Diagrammen verbunden werden. Die Hyperlinks können vom Modellierer selbst oder über das RWI-API auch automatisch gesetzt werden.

In Tabelle 9.2 werden die wichtigsten Navigationsmöglichkeiten aufgelistet, die aus Gründen der Übersicht und Bedienbarkeit von einem Modellierungswerkzeug zur Modellierung eines Business Object Modells automatisch den Modellelementen der Diagramme zugewiesen werden sollten. Jeder Tabelleneintrag hat dabei folgendes Format:
Element@Diagrammtyp1 $\leftrightarrow$ | $\rightarrow$ Element@Diagrammtyp2 + Kommentar
Das bedeutet, daß es eine Navigationsmöglichkeit vom Element wie es in Diagrammtyp1 dargestellt ist auf das aus Sicht des Business Object Modell selbe Diagrammelement im Diagrammtyp2. Sollten die Elemente nicht die gleichen Entitäten sein, wird dies im Kommentar erläutert. Der Zusatz „-Diagramm“ wird aus Gründen der Übersichtlichkeit bei der Bezeichnung der Diagrammtypen weggelassen.

Prinzipiell sind neben diesen noch weitere Navigationsmöglichkeiten denkbar. Hier sollten zunächst einmal nur die grundsätzlichen Navigationspfade aufgezeigt werden. Es gilt jedoch: Ein mehr an Navigationsmöglichkeiten bringt ein mehr an Übersichtlichkeit. Im der prototypischen Umsetzung der Modellierungswerkzeugs, die in Abschnitt 9.7 besprochen wird, sind alle obigen Möglichkeiten realisiert worden.

Navigationsmöglichkeit	Kommentar
BWO[1]@Business-Workflow → BWO@Business-Workflow BWO@Business-Workflow ↔ BWO@Business-Object BWO@Business-Workflow ↔ BWO@Business-Use-Case BAO@Business-Workflow → BAO@Business-Assignment	[1] Subworkflow
BTO@Business-Assignment → BTO@Business-Object BAO@Business-Assignment → BAO@Business-Object	
BWO@Business-Object ↔ BWO@Business-Use-Case BTO@Business-Object ↔ BTO@Business-Assignment BAO@Business-Object ↔ BAO@Business-Organization	
BEO@Business-Organiz. → BEO@Business-Object	

Tabelle 9.2: Wichtige Navigationsmöglichkeiten zwischen den Diagrammen

9.4 Werkzeugunabhängiges Infrastrukturmodell

Soll ein Business Object Modell die im Abschnitt 3.5 aufgeführten Kriterien erfüllen, so muß es unabhängig von technischen Aspekten einer Infrastruktur sein. Auch die Elemente des Modells dürfen dann keine Beziehungen zu Klassen oder Komponenten haben, die keine Abbildung der Geschäftswelt darstellen.

Die Abbildung eines Business Object Modells auf eine Infrastruktur zur Realisierung eines Business Object Systems, erfordert es aber, das Modell um Designentscheidungen und plattformabhängige Generierungsinformationen anzureichern, die die Abbildung des Modells steuern. Dies führt aber einerseits zu einer „Verschmutzung“ des Modells und andererseits wird das Business Object Modell hauptsächlich mithilfe der fünf in Kapitel 7 eingeführten Diagrammtypen modelliert und dargestellt, deren Möglichkeiten zur Ergänzung um infrastruktur- und plattformabhängige Elemente sehr beschränkt ist.

In dieser Arbeit wird als Lösung für dieses Problem in der Architektur der Werkzeugumgebung eine Zwischenschicht zwischen Business Object Modell einerseits und Infrastruktur andererseits eingeführt. Integrationspunkt dieser Zwischenschicht ist ein Modell, das sowohl Business Object Modell als auch technische Erweiterungen für die Abbildung auf die Infrastruktur vorsieht, das sogenannte Infrastrukturmodell.

Das Infrastrukturmodell soll dabei möglichst flexibel für die Einbindung von Erweiterungen einer großen Bandbreite möglicher Zielinfrastrukturen und -Plattformen sein. Auch soll es von bereits existierenden, kommerziellen Werkzeugen bearbeitet und verwaltet werden können. Zur Erfüllung dieser Anforderun-

gen wird ein XML-konformes Infrastrukturmodell vorgeschlagen, das eXtended Business Object Model und dafür eine XML-Beschreibungssprache, die eXtended Business Object Model Definition Language (XBODL) definiert.

Im nächsten Abschnitt sollen die Konzepte von XML kurz dargestellt werden und die Gründe diskutiert, warum XML in dieser Arbeit zur Darstellung des Infrastrukturmodells ausgewählt wurde. Danach wird die XBODL vorgestellt.

9.4.1 XML

Die XML ist eine Sprache zur Darstellung und Definition von semistrukturierten Dokumenttypen und -Elementen. Kernpunkt der XML ist eine Definitionssprache, die die Elemente der Dokumente und ihre syntaktisch korrekte Form beschreibt. Ein Dokumenttyp wird in sogenannten Document Type Definitions (DTD) definiert. Ein Werkzeug, das mit diesem Dokumenttyp arbeiten will, liest die DTD ein und kann so einerseits die Elemente des Dokumenttyps erkennen und andererseits entscheiden, ob ein vorliegendes Dokument syntaktisch korrekt ist.

Das ist ein wesentlicher Vorteil gegenüber anderen Ansätzen, denn die Werkzeuge sind damit unabhängig von einem bestimmten Dokumenttyp und damit für alle XML-konformen Dokumente einsetzbar. Dies ist auch ein Grund dafür, daß es inzwischen eine große Anzahl von kommerziellen und frei verfügbaren XML-Werkzeugen sowie -Komponenten gibt. Gerade die Komponenten ermöglichen es, zwei bestehende Anwendungen um die Möglichkeit zu ergänzen, untereinander Dokumente in einem bestimmten Format auszutauschen. Dazu ist lediglich die nachträgliche Einbindung einer XML-Parser-Komponente und die Definition einer geeigneten DTD notwendig. Gerade der Austausch und die gemeinsame Nutzung von semistrukturierten Dokumenten zwischen verschiedenen Anwendungen ist ein Haupteinsatzgebiet der XML.

Zur Veranschaulichung des XML-Konzepts folgendes Beispiel: Es soll ein Dokumenttyp definiert werden, der den Austausch von eMails zwischen verschiedenen Anwendungen gestattet. Vorausgesetzt die Anwendungen sind bereits um einem XML-Parser ergänzt, muß zunächst eine DTD definiert werden. Im Fall eines eMail-Dokumenttyps soll das Dokument bestehen aus der eMail mit einer Nachrichtennummer, aus der eMail-Adresse des Senders, ein oder mehreren Empfänger-Adressen, einem Thema und der (nicht weiter strukturierten) Nachricht.

Eine DTD für einen solchen Dokumenttyp könnte folgendermaßen aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT email (sender,empfaenger+,thema,nachricht)>
<!ATTLIST email  id CDATA #REQUIRED>

<!ELEMENT sender (#PCDATA)*>
<!ELEMENT emfaenger (#PCDATA)*>
<!ELEMENT thema (#PCDATA)*>
<!ELEMENT nachricht ANY>
```

Es werden die Elemente `email`, `sender`, `empfaeger`, `thema` und `nachricht` definiert, wobei `sender`, `empfaeger`, `thema` und `nachricht` Teile eines `email`-Elements sind. Zudem wurde ein Attribut `id` für das Element `email` definiert. Ein XML-Dokument als Instanz dieses Dokumenttyps könnte dann zum Beispiel folgenden Inhalt haben:

```
<email id="83726637">
  <sender> molterer@in.tum.de </sender>
  <empfaenger> broy@in.tum.de </empfaenger>
  <empfaenger> dippold@in.tum.de </empfaenger>
  <thema> Terminvorschlag </thema>
  <nachricht>
    Sehr geehrter Herr Prof. Broy,
    [...]
  </nachricht>
</email>
```

Für eine umfassende Einführung in die Möglichkeiten von XML wird auf [Meg98] verwiesen. Wichtig für das Infrastrukturmodell auf Basis von XML sind vor allem folgende drei Eigenschaften:

- Die Möglichkeit, semistrukturierte Dokumente zu erlauben. Denn das Infrastrukturmodell sollte flexibel und offen für verschiedene Infrastrukturen und Plattformen sein. Strukturiert werden kann das Business Object Modell und die Zuordnung von Erweiterungen; die Struktur aller möglichen Erweiterungselemente für alle in Frage kommenden Infrastrukturen ist aber nicht sinnvoll. Diese Einträge sollten möglichst frei gestaltbar sein.
- Die Verfügbarkeit von einer großen Anzahl von Werkzeugen und Komponenten zur Nutzung von XML-Dokumenten. Denn die Möglichkeit bereits

bestehende Werkzeuge zu nutzen steigert die Akzeptanz einer Methode. Außerdem wird eine gemeinsame Nutzung des Infrastrukturmodells durch verschiedene Werkzeuge erleichtert, da genügend Komponenten vorhanden sind, diese anzupassen und zu ergänzen.

- Die XML-Dokumente sind nicht in Binärform, sondern lesbar. Das heißt, auch wenn es kein spezielles Werkzeug zur Darstellung des Infrastrukturmodells gibt, genügt ein normaler Texteditor, um das Modell zu lesen und zu bearbeiten.

9.4.2 eXtended Business Object Definition Language

Das Infrastrukturmodell wird durch XML-Dokumente, die den Definitionen des Dokumenttyps eXtended Business Object Modelling Language (XBODL) entsprechen, in der Zwischenschicht der Werkzeugumgebung verwaltet. In diesem Abschnitt werden die Konzepte und Elemente des Dokumenttyps vorgestellt. Die komplette XBODL-DTD ist in Anhang C dargestellt.

Die oben bereits dargestellten funktionalen Anforderungen an die XBODL sind:

- Vollständige Beschreibung eines Business Object Modells
- Erweiterung des Modells um Infrastrukturelemente
- Keine Einschränkung der XBODL auf eine Infrastruktur

Die XBODL erfüllt die Anforderungen durch die Aufteilung des Dokumentes in drei Abschnitte: Elemente, Referenzen und Erweiterung, sowie durch die Einführung von eindeutigen Identifikatoren für jedes Modellelement und durch frei strukturierbare Definitionen von Erweiterungsbeschreibungen.

Die drei Abschnitte dienen zur besseren und übersichtlicheren Strukturierung des Dokuments. In Abbildung ist die Struktur und die Elemente der XBODL in der Diagrammart „XML DTD“ des Modellierungswerkzeugs Together dargestellt. Die Namen der Kästen stellen dabei die zu definierenden Elemente der DTD dar, die Pfeile die möglichen „Ist-enthalten-in“ Relationen. Die Elemente in den Kästen sind die möglichen Attribute des Elements.

Im Dokumentabschnitt `elements` werden alle Entitäten des Business Object Modellelements definiert: Domänen (`domains`), Business Entity Objects (`beo`),

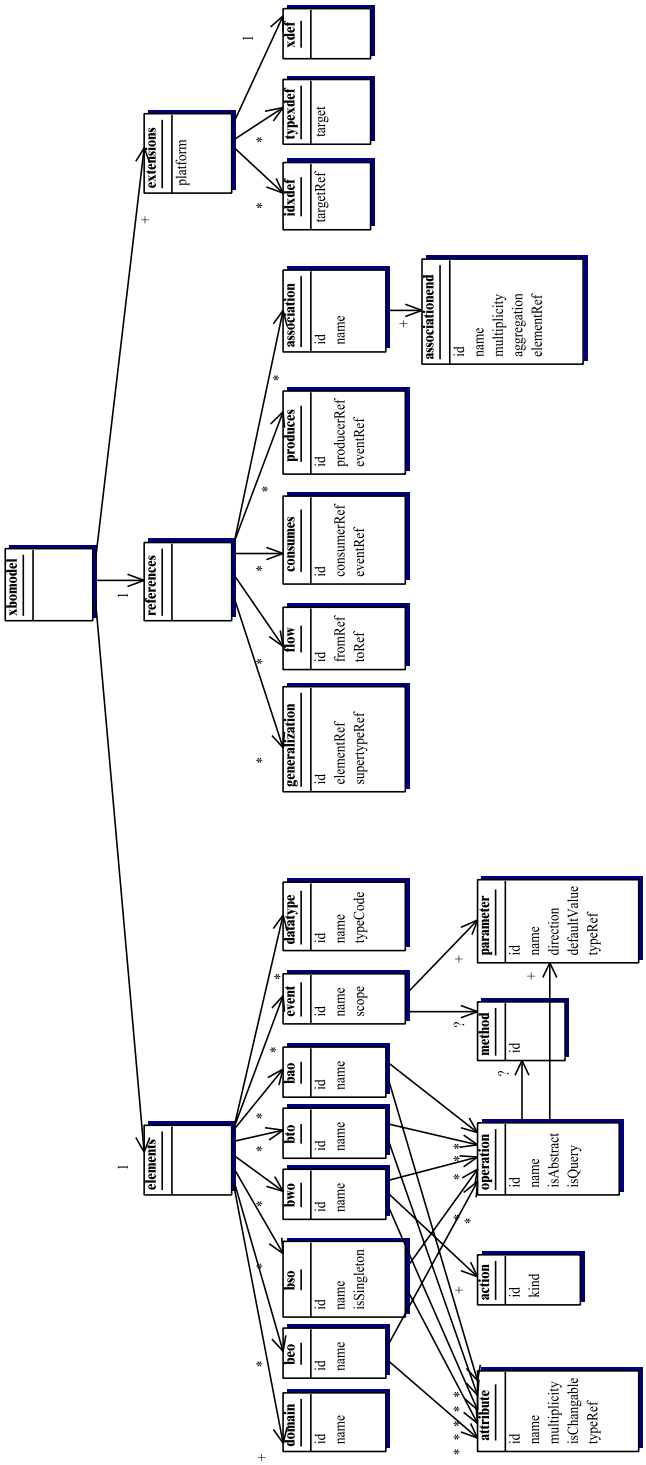

Abbildung 9.3: Struktur und Elemente der XBODL in der Übersicht

Business Service Objects (`bso`), Business Workflow Objects (`bwo`), Business Task Objects (`bto`), Business Actor Objects (`bao`) und Events (`event`) sowie Datentypen (`datatype`). Zudem können die Definitionen der Business Objects und Events Attribute (`attribute`), Operationen (`operation`) und Methodenrümpfe (`method`) enthalten.

Im Abschnitt `references` werden alle möglichen Beziehungen zwischen den einzelnen Elementen definiert: Assoziationen (`association`), Generalisierungsbeziehungen (`generalization`), Kontrollflußbeziehungen (`flow`) und die Eventkanal-Beziehungen (`consumes` und `produces`).

Damit eine Trennung der Entitäten von den Beziehungen überhaupt möglich ist, wurde für jedes Element der XBODL ein eindeutiger Identifikator (`id`) als Attribut vorgesehen. Alle Attribute die basierend auf diesen Identifikatoren eine Beziehung zwischen zwei XBODL-Elementen herstellen, haben ein oder mehrere Attribute, deren Namen mit „Ref“ enden.

Auf diese Weise wird auch die Anknüpfung der Erweiterungselemente für die Infrastrukturabbildung an die Business Object Elemente realisiert. Die Erweiterungen befinden sich in ein oder mehreren Abschnitten `extensions` des XBODL-Dokuments. Für jede Zielplattform oder -Infrastruktur kann ein eigener `extensions`-Abschnitt angegeben werden, der durch das Attribut `platform` von den anderen Erweiterungsabschnitten unterschieden wird.

Das Erweiterungskonzept der XBODL sieht drei Möglichkeiten zur Anbindung von Abbildungsanweisungen an das eigentliche Business Object Modell der Abschnitte `elements` und `references` vor. Zum Einen durch das Element `idxdef`, das eine Zuweisung an ein konkretes Modellelement mit dem Identifikator `targetRef` ermöglicht. Zum Anderen durch das Modellelement `typexdef`, das eine Zuweisung an einen Modellelement-Typ, also zum Beispiel „beo“ oder „event“ zuläßt. Und schließlich das Element `xdef`, das eine globale Abbildungsanweisung für das gesamte Modell darstellt.

Gültiger Inhalt von `idxdef`, `targetRef` und `xdef` sind beliebige, nicht weiter strukturierte Texte. Damit ist die Anforderung erfüllt, dem Entwickler eine möglichst flexible Möglichkeit der Spezifikation von Erweiterungsanweisungen für verschiedene Abbildungsszenarien zu bieten.

Mit der XBODL hat der Entwickler eines Business Object System nunmehr die Möglichkeit, ein Business Object Modell – das ja dem Business Object Paradigma folgend unabhängig ist von Anwendungen, Plattformen und Business Object Infrastrukturen – um Elemente zu erweitern, die eine Abbildung des Modells auf

eine Business Object Infrastruktur steuern. Wie eine Abbildung auf Basis des Enterprise JavaBeans Frameworks aussehen kann, wird im nächsten Abschnitt zur untersten Schicht der Architektur der Werkzeugumgebung gezeigt.

9.5 Abbildung auf die Infrastruktur

Abschluß eines unternehmensweiten Vorgehens auf Basis von Business Objects ist die Abbildung des Business Object Modells auf eine geeignete Infrastruktur. Dabei wird weiter nach dem Prinzip des Convergent Engineering vorgegangen: Ziel sind offene, unternehmensweite Lösungen aus Komponenten, die die Elemente der Geschäftswelt widerspiegeln.

In der untersten Schicht der Werkzeugumgebung müssen dem Entwickler vor allem zwei Hilfsmittel zur Verfügung stehen, um diese Lösungen zu realisieren:

- Ein Framework, auf dessen Komponenten die Elemente eines Business Object Systems aufsetzen und betrieben werden können.
- Ein Generierungs-Werkzeug, das auf Basis eines Infrastrukturmodells die Elemente des Business Object Systems auf die Framework-Elemente automatisiert abbildet.

In diesem Abschnitt werden zunächst die Anforderungen an eine Infrastruktur zum Betrieb von Business Object Systemen diskutiert. Danach wird das Enterprise JavaBeans-Framework unter dem Gesichtspunkt vorgestellt, ob und inwieweit es sich als eine mögliche Business Object Infrastruktur eignet. Dem folgt abschließend ein Konzept zur Abbildung des Business Object Modells auf Enterprise JavaBeans.

9.5.1 Anforderungen an eine geeignete Infrastruktur

Eine geeignete Infrastruktur für Business Object Systemen hat die Aufgabe, ein offenes, unternehmensweites System das passive, datenorientierte Business Entity Objects wie auch aktive, vorgangsorientierte Business Workflow Objects enthält, unterstützen und betreiben zu können. Dabei soll die Abbildung des Modells auf die Infrastruktur möglichst automatisch durch ein Generierungs-Werkzeug erfolgen. Daraus ergeben sich mehrere Anforderungen:

- Die Infrastruktur sollte die Entwicklung von feingranularen, verteilten und persistenten Komponenten ermöglichen. Verteilte Komponenten deshalb, weil sich für die Clients des Systems die Architektur des Systems so darstellen soll, wie die Architektur des Business Objects Modells. Diese wiederum entspricht der Architektur der Elemente der Geschäftswelt. Feingranular deshalb, weil auch die Elemente der Geschäftswelt in der Regel nur ein einzelnes Konzept darstellen. Persistent, weil die Business Objects in der Regel einen Zustand haben, der unabhängig von der Betriebszeit des Komponentensystems ist.

- Die Komponenten sollten möglichst unabhängig von den Infrastrukturdiensten wie der Persistenzhaltung sein. Je abhängiger die Komponenten von den Diensten sind, je schwieriger ist es eine automatisierte Abbildung zu ermöglichen.

- Die Infrastruktur sollte aktive Komponenten zulassen, die von sich aus Zustände im Business Object System verändern können. Aktiv deshalb, da auch arbeitsteilige Vorgänge unterstützt werden sollen, die Komponenten benötigt, die aktiv für den Fortgang und die Ablaufsteuerung von Vorgängen zuständig sind.

- Die Infrastruktur sollte erweiterbar für das Business Object Eventmodell sein. Attributänderungen und Methodenaufrufe müssen nebenläufig Methoden anderer Komponenten anstoßen können.

- Die Infrastruktur sollte sich deklarativ konfigurieren und anpassen lassen. Dadurch ist es dem Entwickler möglich, entsprechende Anweisungen in das Infrastrukturmodell zu übernehmen. Dies wiederum erlaubt eine weitgehend automatische Abbildung des Business Object Modells und der zusätzlichen Erweiterungen für das Framework auf die Infrastruktur.

Diese Anforderungen werden vom Enterprise-JavaBeans-Framework weitestgehend erfüllt. Im nächsten Abschnitt werden die passenden wie fehlenden Eigenschaften des Frameworks als Basis für Business Object Systeme diskutiert.

9.5.2 Enterprise JavaBeans

Das Enterprise JavaBeans (EJB) Framework der Firma Sun wurde 1998 mit dem Ziel entwickelt, ein Server-seitiges Komponentenmodell für die Programmiersprache Java zur Verfügung zu stellen. Der große Akzeptanz des Frameworks

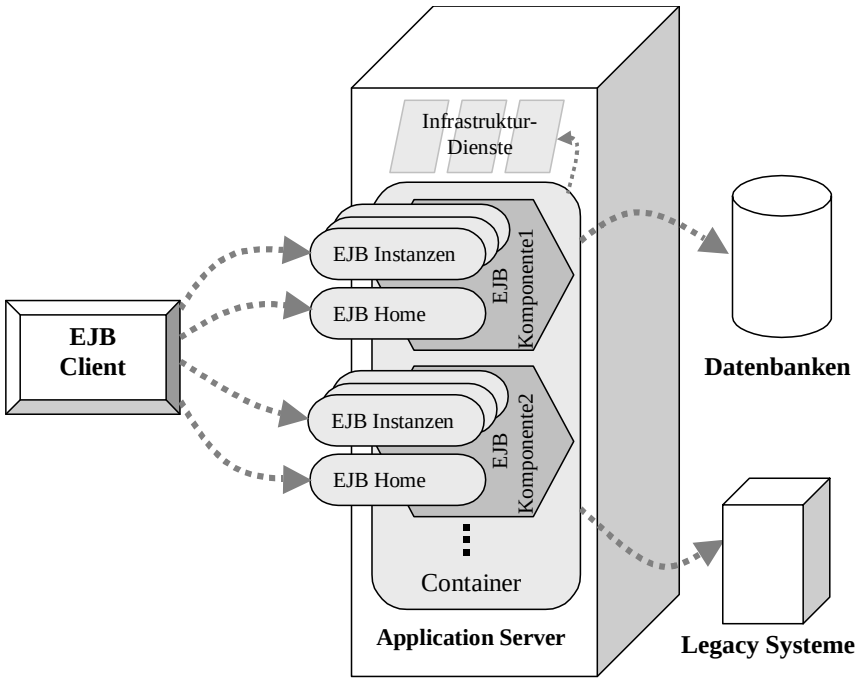

Abbildung 9.4: Architektur und Elemente der J2EE in der Übersicht

ist auf eine frühzeitige Einbindung von Anwendern in seine Entstehung, Weiterentwicklung und kundenorientierte Verbesserung, die kostenlose Verfügbarkeit und nicht zuletzt durch die inzwischen zahlreich vorhandenen kommerziellen Produkte zurückzuführen.

Die Entwicklung des Frameworks war für Sun verbunden der Idee, eine Standardplattform bestehend aus einem Architekturkonzept und verschiedenen Frameworks zur Entwicklung komponentenbasierter Informationssysteme – die Java 2 Enterprise Edition (J2EE) – bereitzustellen. EJBs sind ein Teil dieser Plattform. Eine umfassende Darstellung dieser Standardarchitektur und seiner Frameworks findet sich in [Rom99].

In Abbildung 9.4 werden die wesentlichen Elemente einer Architektur auf Basis der J2EE mit EJBs dargestellt. Kernelemente sind die EJB-Komponenten. Im EJB-Komponentenmodell sind verschiedenen Arten von EJB-Komponenten vorgesehen:

- Entity Beans zur Klassifizierung persistenter Instanzen einer Kernentität der Anwendung, zum Beispiel „Kunde“ oder „Auftrag“.
- Session Beans zur Bereitstellung von nicht-persistenten, fachlichen Diensten

auf diesen Kernentitäten, wie zum Beispiel „Kundenverwaltung“. Dabei gibt es die Unterscheidung zwischen Stateless oder Stateful Session Beans. Stateful Session Beans behalten von Client- zum nächsten Clientzugriff ihren Zustand, Stateless Session Beans haben bei jedem Zugriff den gleichen Anfangszustand.

- Message-driven Beans zur asynchronen Aktivierung von Entity oder Session Beans beim Auftreten einer Nachricht in einem zuvor abonnierten Nachrichtkanal.

Für jede dieser Komponente können zur Laufzeit des Systems mehrere Instanzen existieren, die von einem Client mithilfe eines eindeutigen Bezeichner (Handles) unterschieden und genutzt werden können. Daneben gibt es noch ein sogenanntes Home-Interface, das Operationen auf allen Instanzen der EJB-Komponente, wie zum Beispiel eine Suchanfrage, bündelt.

Die Verwaltung der Komponenten übernimmt ein sogenannter Container. Seine Hauptaufgabe ist es, einen definierten Lebenszyklus und die Bereitstellung von Ressourcen für die Instanzen der EJB-Komponenten zu gewährleisten. Außerdem ist er zuständig für die Speicherung des Zustandes von Entity Beans in Datenbanken oder anderen Systemen. Für jeden Datenbank- bzw. Systemtyp gibt es dabei jeweils einen Containertyp. Die Container ihrerseits sind Bestandteil eines Applikationsservers. Er bietet verschiedene Infrastrukturdienste, wie Autorisierung, Nachrichtenmanagement, Transaktionsmanagement oder einen Namensdienst zur Nutzung durch die Container an.

Die EJBs selbst nutzen diese Infrastrukturdienste nicht direkt. Das Grundlegende am Container-Konzept läßt sich durch das Motto „Don’t call us, we call you!“ zusammenfassen. Der Container kümmert sich für die EJBs um die Nutzung der Dienste. Im Idealfall enthalten EJBs nur Attribute und Methoden zur Darstellung und Manipulation des fachlichen Anwendungsmodells. Für Clients – das können andere EJBs oder Anwendungs-Clients sein – ist der Container aber nicht sichtbar. Die Nachrichten zur Erzeugung, Löschung und Nutzung der EJBs durch die Clients werden – vollkommen transparent für die Clients – vom Container abgefangen, die notwendigen Dienstaufrufe getätigt und dann auf die eigentlichen EJB-Instanzen umgesetzt. Die EJB-Komponenten verhalten sich für äußere wie innere Clients wie verteilte Komponenten.

Die Entwicklung von EJBs ist in mehrere Schritte aufgeteilt: Zunächst werden die EJB-Komponenten unabhängig voneinander entwickelt. Jede dieser Komponenten wird dann in einen Archiv zusammen mit einem sogenannten Deployment Descriptor abgelegt. Mit Hilfe dieses Deployment Descriptor kann ein Entwick-

ler deklarativ auf Basis eines XML-Dokumentes die Eigenschaften einer EJB-Komponente anpassen. Zum Beispiel in welcher Datenbank sie gespeichert werden soll oder unter welchem Namen sie im Namensdienst eingetragen werden soll. Die EJB-Archive können dann zu einem größeren Archiv zusammengefaßt werden, das in der Regel alle EJBs eines Anwendungsmodells enthält. Auch hier hat der Entwickler die Möglichkeit, mit einem Deployment Descriptor die Eigenschaften des Komponentensystems, zum Beispiel die Festlegung der Beziehungen zwischen den EJB-Komponenten, deklarativ zu beschreiben. Schließlich wird dieses Anwendungsarchiv auf einem Applikationsserver durch das sogenannte „Deployment“ in Betrieb genommen.

Im vorigen Abschnitt wurden Anforderungen an eine geeignete Infrastruktur für Business Object Systeme definiert. Das EJB-Framework erfüllt dabei die meisten dieser Anforderungen:

- Das EJB-Framework ermöglicht die Entwicklung von feingranularen verteilten und persistenten Komponenten.
- Die Container-Architektur erlaubt es, die Komponenten unabhängig von den Infrastrukturdiensten zu implementieren.
- Durch Deployment Deskriptoren können EJB-Komponenten und -Systeme deklarativ konfiguriert und angepaßt werden. Die Deployment Descriptoren können in das Infrastrukturmodell aufgenommen werden und gewährleisten so eine automatische Abbildung des Business Object Modells auf EJBs.

Allerdings gibt es auch Eigenschaften, die dem EJB-Framework zur Erfüllung aller genannten Anforderungen fehlen:

- Das EJB-Framework läßt keine aktive Komponenten zu. Im Gegenteil: In der Spezifikation [SUN99a] heißt es: „An enterprise Bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop or resume a thread; or change a threads priority or name. The enterprise bean must not attempt to manage thread groups.“. Dies schränkt die Möglichkeit ein Business Actors als aktive Komponenten zu Steuerung eines Workflows zu realisieren. Es gibt jedoch auf Basis der Message-driven Beans die Möglichkeit, diese Eigenschaft nachzubilden. Ein solches Konzept wird in Abschnitt 9.5.3.3 vorgestellt.
- Das EJB-Framework besitzt kein Eventmodell. Wie eine Nachbildung eines solchen Modells zur Realisierung der Eventkanäle des Business Object Mo-

dells durch Message-driven Beans aussehen kann, wird in Abschnitt 9.5.3.2 skizziert.

Im folgenden Abschnitt wird ein Konzept vorgestellt, wie die Elemente des Business Object Modells auf das EJB-Framework abgebildet werden können.

9.5.3 Abbildung der Modellelemente

Das EJB-Framework erfüllt wie oben diskutiert die Anforderungen an eine Business Object Infrastruktur nicht vollständig. In diesem Abschnitt wird eine mögliche Abbildung der Elemente des Business Object Modells auf die EJB-Konzepte vorgestellt.

Zunächst die Abbildungen von Business Entity und Business Service Objects sowie deren Assoziationen, die direkt erfolgen können, dann ein Vorschlag zur Nachbildung des Eventmodells und schließlich die etwas umfangreichere Abbildung der Workflow-Elemente Business Workflow Object, Business Actor Object und Business Task Object.

9.5.3.1 Business Entity und Service Objects

Business Entity Objects und Business Service Objects sowie deren Attribute, Operationen und Assoziationen können direkt auf entsprechende Elemente des EJB-Komponentenmodells abgebildet werden:

Business Entity Objects werden auf Entity JavaBeans abgebildet, die statischen Operationen werden in das Home Interface aufgenommen, die nicht-statischen in das Entity Interface. Für jedes Attribut werden Getter- und Setteroperationen in das Entity Interface aufgenommen. Die Attribute selbst sollen von außen nicht zugreifbar sein. Die Methoden werden in die entsprechenden Methoden des EJB Objects übernommen.

Assoziationen zwischen Business Entity Objects werden auf Container-managed Relationships übertragen. Container-managed Relationships können mithilfe des `ejb-relationship`- und `ejb-relationship-role`-Elements des Deployment Descriptors definiert werden. Die vorgegebenen Möglichkeiten 1:1, 1:n und n:n Beziehungen sowie die Angabe von Aggregations- und Kompositionsbeziehungen sind ausreichend für alle denkbaren Assoziationsbeziehungen im Business Object Modell.

Für jedes Attribut des Business Entity Objects müssen durch Einträge im Deployment Descriptor als Container-managed Persistency Felder deklariert werden. Sie werden dann vom Container automatisch persistent gehalten.

Business Service Objects können auf Stateful Session Beans abgebildet werden. Mit den Attributen und Operationen wird wie bei den Business Entity Objects verfahren.

Die Methoden der Business Objects sind, wie in vorigen Kapiteln erwähnt, in dieser Arbeit als Subset der Programmiersprache Java vorgesehen. Folgende Einschränkungen gelten:

- Es dürfen nur die Packages java.lang, java.math, java.text, java.util verwendet werden.
- Die Definition von Inline-Klassen ist nicht erlaubt.
- Die folgenden Klassen des Packages java.lang dürfen nicht benutzt werden: Class, ClassLoader, Compiler, InheritableThreadLocal, Process, Runtime, RuntimePermission, SecurityManager, System, Thread, ThreadGroup, ThreadLocal, Throwable.
- Das Erzeugen einer Exception mit `throw()` ist nicht gestattet.
- Es dürfen keine Threads gestartet werden.

Die Zugriffe auf andere Business Objects werden als normaler Zugriff auf Klassen und Objekte spezifiziert und müssen bei der Generierung auf entfernte Zugriffe auf EJB-Komponenten umgesetzt werden.

9.5.3.2 Events

Das EJB-Framework unterstützt keine Eventmodell. Das Business Object Modell sieht es vor, Eventkanäle zwischen Business Objects zu ermöglichen, über die Events verschickt werden, wenn sich ein Attribut ändert oder eine Methode aufgerufen wird. Ein solches Eventmodell setzt zudem voraus, daß der Event asynchron getriggert wird. Ein asynchroner Aufruf von EJB-Komponenten ist aber im Komponentenmodell nicht vorgesehen.

Um das Eventmodell trotzdem auf EJBs abbilden zu können, wird eine Nachrichtenbasierte Lösung auf Basis der Message-driven Beans vorgeschlagen. Für jeden

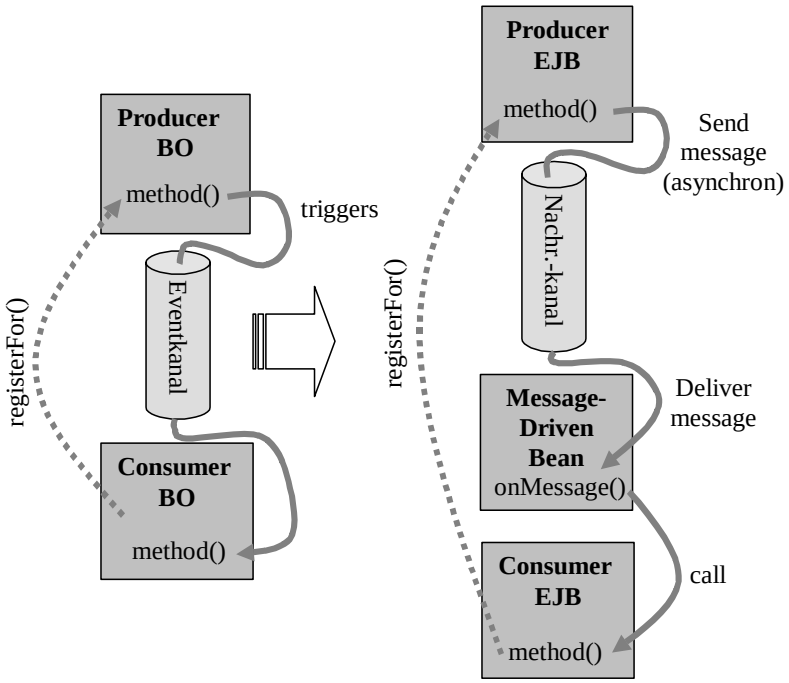

Abbildung 9.5: Die Umsetzung des Eventmodells auf das EJB-Framework

Eventkanal wird ein Nachrichtenkanal im Deployment Descriptor definiert sowie ein Message-driven Bean das auf Nachrichten in diesem Kanal wartet.

Bei der Generierung der EJB-Komponenten wird in jede Methode einer Producer-Operation eines Business Objects ein zusätzliches Code-Fragment eingefügt, daß ein Event-Object erzeugt und es an den für den Eventkanal vorgesehenen Nachrichtenkanal als Nachricht schickt. Bei einem Producer-Attribut wird ein solches Code-Fragment am Ende der Setter-Methode eingefügt.

Die Nachricht wird dann asynchron vom Message-driven Bean empfangen, indem die Nachrichten-Middleware die Methode `onMessage()` des Message-driven Bean aufruft. Das Bean fungiert als Event-Dispatcher und ruft seinerseits die Consumer-Methode des für den Event registrierten Business Object auf und übergibt dabei das Event-Object. Abbildung 9.5 zeigt das Umsetzungskonzept in der Übersicht.

Dieses Verfahren hat seine Schwächen: Nachrichten erzeugen, senden und weitergeben sind sehr teure Operationen. Deshalb sollten Events im Business Object Modell nur dann angewendet werden, wenn die Frequenz der Event-Signalisierung für einen Eventkanal niedrig ist.

9.5.3.3 Business Workflow, Task und Actor Objects

Die Umsetzung von Business Workflow Objects und die damit zusammenhängenden Business Objects ist auf das EJB-Framework nicht direkt möglich. Ein EJB-Container kann auf eine Vielzahl verschiedenster Dienste wie Persistenzmanagement, Transaktionsmanagement, Life-Cycle-Dienste oder Namensdienste zurückgreifen. Ein Workflow-Managementdienst ist jedoch nicht darunter.

Deshalb muß ein solcher Dienst nachgebildet werden. Die Hauptaufgaben eines solchen Dienstes wurden bereits in Kapitel 4 vorgestellt. Die wichtigsten sind: Verwaltung und Instantiierung der Workflow-Schema, Steuerung des Kontrollflusses und Zuweisung von Aufgaben zu den Akteuren (Workpool) sowie zu den konkreten Bearbeiter (Worklist).

Das nachfolgend vorgeschlagene Konzept beruht auf dem Business Object Eventmodell und gestattet eine Ausführung von Workflows ohne zentrale Steuerungskomponente.

Grundsätzlich gilt: Business Workflow Objects, Business Task Objects und Business Actor Objects werden wie Business Entity Objects auf Entity Beans abgebildet. Auch die Action-Elemente zur bedingten Verzweigung und Zusammenführung einer parallelen Ausführung des Business Object Workflows werden jeweils auf Entity Beans umgesetzt.

Die im Kapitel 6 definierten Framework-Operationen `resume()`, `terminate()`, `abort()`, `suspend()` und `start()` bei den Business Workflow Objects und `resume()`, `terminate()`, `abort()`, `suspend()`, `start()` und `complete()` bei den Business Task Objects werden durch Eventkanäle miteinander verbunden.

Die Workflow-Schema liegen im Falle des Business Objects Modells als Business Workflow Objects vor. Sie enthalten Business Task Objects, Business Actor Objects, Actions und Flow-Elemente zu Spezifikation des Kontrollflusses.

Zur Instantiierung eines solchen Schemas des auf EJBs abgebildeten Business Workflow Objects wird die Methode `create()` aufgerufen. Hier müssen während der Generierung die in diesem Workflow enthaltenen Elemente als auch ihre Kontrollflußverbindungen untereinander instantiiert werden. Diese Methode führt folgende Anweisungen aus:

Die Instantiierung der enthaltenen Elemente erfolgt durch Aufruf ihrer `create()`-Methoden. Bei Subworkflows wird so die Instantiierung rekursiv fortgesetzt.

Ein Kontrollfluß-Übergang zwischen zwei Elementen wird durch einen Event-Kanal ersetzt: Die `start()`-Methode des Elements, das den Kontrollfluß übernimmt, wird getriggert, wenn die `terminate()`, `abort()` oder `complete()`-Methode des anderen Elements ausgeführt wird.

Bei einem Startzustand wird die `start()`-Methode des auf den Startzustand folgenden Elements getriggert, wenn die `start()`-Methode des Workflows aufgerufen wird. Bei einem Endzustand wird die `terminate()`-Methode des Workflows durch die `terminate()`, `abort()` oder `complete()`-Methode des dem Endzustand vorausgehenden Elements getriggert.

Die Zuweisung von Akteuren und Bearbeitern erfolgt ebenfalls mithilfe des Eventkonzepts: Die Business Actor Objects sind als Entity Beans abgebildet und zwar als Singletons. Das heißt, pro Entity Bean gibt es genau eine Instanz. Es gibt zwei Arten von Business Actor Objects, die im Business-Organization-Diagramm spezifiziert werden: Roles und Actors. Roles sind zuständig für die Workpool-Verwaltung, Actors sind zuständig für die Worklist-Verwaltung.

Die `assign()`-Frameworkmethode einer Role-Instanz wird durch Eventkanäle mit der `start()`-Methode der in allen Workflow zugeordneten Business Task Objects verbunden. Wird die `assign()`-Methode dadurch getriggert, so wird das Business Task Object in den Workpool des Business Actor Objects eingetragen. Zusätzlich registriert sich das Business Actor Object mit der Methode `deassign()` für die `execute()`-Methode des Business Task Objects.

Die Actor-Instanzen sind die Anknüpfungspunkte für die Business Object Anwendungen. Für jeden Benutzer des Systems gibt es eine Actor-Instanz. Eine Business Object Anwendung zeigt dem Benutzer alle Aufgaben an, die in den Workpool-Listen derjenigen Business Actor Objects (Role), die mit „seinem“ Business Actor Object (Actor) im Business-Organization-Diagramm verknüpft wurden. Wählt er eine Aufgabe zur Bearbeitung aus, so wird von der Business Object Anwendung die Methode `execute()` des Business Task Objects aufgerufen und die Aufgabe in die Worklist des Business Actor Objects übernommen.

Dies triggert die oben erwähnte `deassign()`, in der die nun einem konkreten Bearbeiter zugewiesene Aufgabe wieder aus ihrem Workpool entfernt. Business Object Anwendungen können sich ebenfalls bei den Methoden `assign()` und `deassign()` einer Role-Instanz registrieren und bei einer Änderung des Workpools die Darstellung der Workpool-Liste in der Benutzeroberfläche updaten.

Hat der Benutzer schließlich die Aufgabe bearbeitet oder wird sie abgebrochen, so wird von der Business Object Anwendung die Methode `complete()` be-

ziehungsweise `abort()` aufgerufen. Damit geht der Kontrollfluß an das darauf folgende Element dessen `start()`-Methode dadurch getriggert wird.

Mit Hilfe des Eventkonzept läßt sich so ein vollständig objektorientierter und nur auf den Business Objects basierender arbeitsteiliger Vorgang realisierten. Dadurch das Events verwendet werden, sind die einzelnen Business Objects immer noch unabhängig voneinander und können zum Beispiel in einem anderen Workflow wiederverwendet werden. Allerdings stellt dieses Vorgehen einen hohen Anspruch an das verwendete Generierungs-Werkzeug, das auf Basis des Business Object Modells nicht nur alle Workflow-Elemente auf Enterprise JavaBeans abbilden, sondern auch eine komplexe Eventkanal-Architektur zwischen den EJBs etablieren muß.

9.6 Prototypische Umsetzung

Im Rahmen einer Diplomarbeit [Frö01] wurde das in diesem Kapitel vorgestellte Werkzeugkonzept zum Teil prototypisch umgesetzt. Das Werkzeug Together wurde durch ein Modul zur Generierung der Enterprise JavaBeans für das Business Object Modell erweitert.

Im Gegensatz zu der hier vorgeschlagenen Lösung wurde der Workflow-Anteil des Modells dabei auf die Elemente des Weblogic Process Integrators (WLPI) abgebildet. Der WLPI ist ein auf Enterprise JavaBeans basierendes Workflow-Management-System. Eine Umsetzung der Workflows so wie es in Abschnitt 9.5 vorgeschlagen wird, konnte in der Diplomarbeit noch nicht verwirklicht werden, da zu diesem Zeitpunkt noch keine Implementierung eines Applikationsservers auf dem Markt war, der Message-driven Beans unterstützt hätte. Diese sind aber unverzichtbarer Bestandteil des Eventkonzepts und damit der Abbildung von Business Object Workflows auf Enterprise JavaBeans.

Fallbeispiel für die Umsetzung war der bereits in Kapitel 7 als durchgängiges Beispiel verwendete Anwendungsfall „Konstruktionsmaßnahme“. Für diesen Anwendungsfall wurden zwei Business Workflow Objects, vier Business Actor Objects, zwölf Business Task Objects, sechs Business Session Objects und die beiden Business Entity Objects „Konstruktionsmaßnahme“ und „Arbeitsfolge“ durch Business-Use-Case-, Business-Assignment-, Business-Workflow-, Business-Object- und Business-Organization-Diagramme modelliert und mithilfe eines auf Basis des RWI-APIs erstellten Generatormoduls erfolgreich auf das EJB-Framework und den WLPI abgebildet. Die Abbildungen 9.6 und 9.7 zeigen

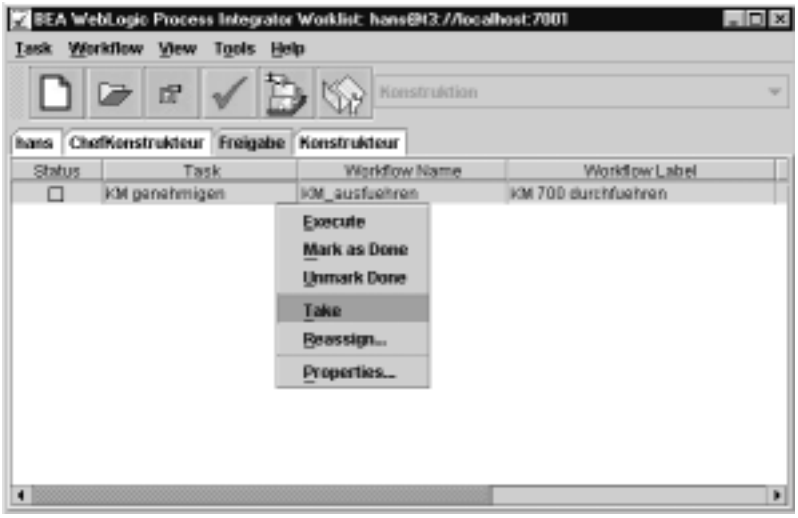

Abbildung 9.6: Screenshot der Business Object Anwendung (aus [Frö01])

den entstandenden Prototypen in Form von zwei Business Object Anwendungen des generierten Business Object Systems „Konstruktionsmaßnahme“.

Eine umfassende Darstellung des Generators dieser prototypischen Umsetzung und das komplette Business Object Modell „Konstruktionsmaßnahme“ finden sich in [Frö01] .

9.7 Zusammenfassung

Die modellbasierte Entwicklung von Softwaresystemen wird heute vorwiegend in einer heterogenen und wenig integrierten Werkzeugumgebung durchgeführt. Dem Entwickler stehen Werkzeuge zur Geschäftsprozeß-, zur Workflow- und zur objektorientierten Modellierung zur Verfügung. Dazu kommen Programmiersprachen-, Datenbankdefinitions- und Codegenerierungs-Werkzeuge. Die Werkzeuge sind vor allem deshalb wenig integriert, weil es zwischen den Modellen und Produkten, die von ihnen bearbeitet und erzeugt werden können, konzeptionelle Unterschiede und über die Phasen der Entwicklung gesehen Brüche gibt.

In dieser Arbeit wird das Business Object Metamodell als Basis für die Beschreibungstechniken, das Vorgehensmodell und die Business Object Systeme selbst vorgestellt. Durch dieses Metamodell werden Unterschiede und Brüche zwischen den Modellen im Sinne eines Convergent Engineerings vermieden. Jedoch ohne eine integrierte Werkzeugunterstützung, die eines korrektes und konsistentes

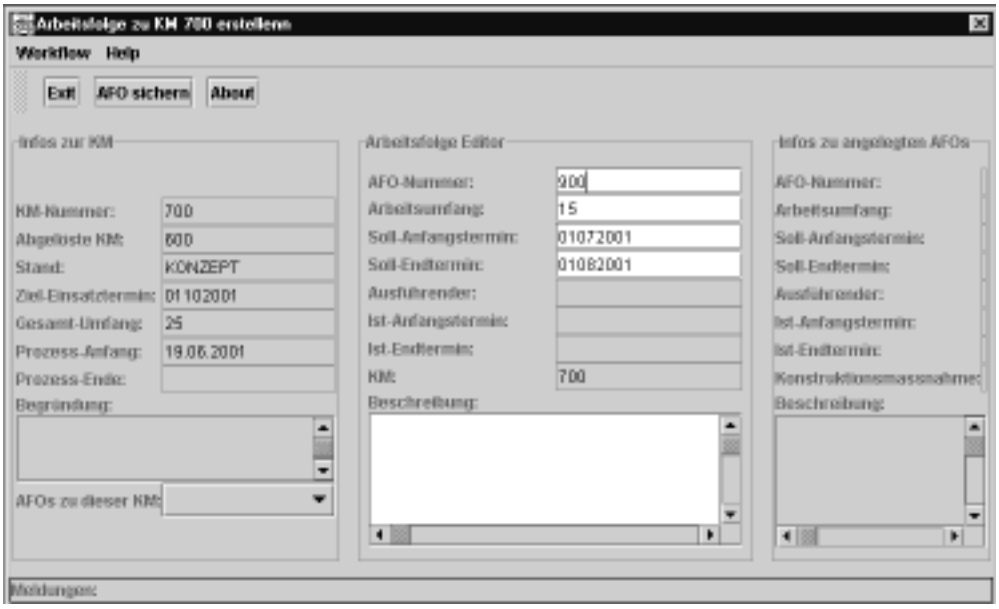

Abbildung 9.7: Screenshot der Business Object Anwendung (aus [Frö01]

Business Object Modell und seine automatischen Generierung und Abbildung auf eine Infrastruktur ermöglicht, ist eine Realisierung von Business Object Systemen nicht möglich.

Das in diesem Kapitel vorgestellte, integrierte Werkzeugumgebung besteht aus drei Schichten. In der obersten Schicht werden die Business Object Modelle erstellt und modelliert. Für diese Schicht wird eine Konzeption für ein UML-basierten Modellierungswerkzeug auf Basis des UML-Tools Together zur Darstellung und Bearbeitung der verschiedenen Sichten auf das Business Object Modells vorgestellt.

In der mittleren Schicht befinden sich Werkzeuge zur Manipulation eines werkzeugunabhängiges Infrastrukturmodells als Zwischendarstellung und Weiterbearbeitung des Business Object Modells zur deklarativen Erweiterung des Modells um technische Aspekte. Eine konkrete Form des Infrastrukturmodells wird durch die Definition der eXtended Business Object Description Language (XBODL), einer XML-Instanz, bereitgestellt.

In der untersten Schicht schließlich befinden sich Werkzeuge und Frameworks zur Abbildung des Infrastrukturmodells auf eine geeignete Infrastruktur – in der Arbeit wird beispielhaft eine Abbildung auf das Enterprise JavaBeans-Framework vorgestellt.

Am Schluß des Kapitels wurde die Tragfähigkeit des Konzepts durch eine Realisierung einer im Rahmen einer Diplomarbeit [Frö01] entstandenen prototypischen Werkzeugumgebung gezeigt.

Mit dem Konzept für eine integrierte Werkzeugumgebung ist das Framework zur Entwicklung unternehmensweiter Anwendungen auf Basis von Business Object vollständig beschrieben. Natürlich gibt es noch Raum für weitere, optionale Frameworkkomponenten. Beispiele für solche Komponenten werden in der Zusammenfassung und Ausblick der Arbeit im nächsten Kapitel diskutiert.

Kapitel 10

Zusammenfassung und Ausblick

Kundenorientierung, Flexibilität und eine globale Ausrichtung sind wesentliche Voraussetzungen für den Erfolg heutiger Unternehmen. Um diese Eigenschaften zu erreichen, müssen funktionsorientierte, segmentierte Prozeßstrukturen neu geordnet werden. Dieses Ziel verfolgt das Business Process Reengineering. Business Process Reengineering steht für eine fundamentale Neuausrichtung der Unternehmensstrukturen auf Kunden-getriebene Geschäftsprozesse. Die traditionellen organisatorischen Grenzen werden zugunsten einer Fokussierung auf diese Kerngeschäftsprozesse aufgelöst.

Die meisten Bereiche und Funktionen von Unternehmen werden bereits unterstützt durch Informationstechnologie. Soll ein Business Process Reengineering zu einer umfassenden Änderung der Unternehmenstrukturen führen, so muß auch die Softwarelandschaft eines Unternehmens neu strukturiert werden. Denn häufig sind die in langen Jahren nach und nach entstandenen Softwaresysteme nur ein Abbild der funktionsorientierten und segmentierten Prozeßstrukturen. Isolierte Systeme und eine heterogene Softwarelandschaft bilden in vielen Unternehmen deshalb die Basis für die Arbeitsabläufe.

Ein Business Reengineering bedeutet also mehr als die bloße Umstellung und Neuordnung der Prozesse. Es stellt auch neue Anforderungen an den Aufbau und die Verknüpfung von Informationssystemen in Unternehmen. So müssen Informationssysteme der Ausrichtung auf unternehmensweite Geschäftsprozesse folgen und nicht umgekehrt. Doch konventionelle Architekturen und Infrastrukturen sind ausgerichtet auf die Realisierung von eigenständigen Systemen für einen abgegrenzten Anwendungsbereich. Es fehlt ein Konzept für eine unternehmensweite, Geschäftsprozeß-orientierte Gestaltung.

Ursachen für die isolierten und auf einzelne Funktionen ausgerichteten Systeme sind aber auch die heutigen Software-Entwicklungsmethoden: Sie sind hauptsächlich mit dem Gedanken entworfen worden, Methoden für die Softwareindustrie bereitzustellen, also für Firmen, die System nach System für unterschiedliche Kunden und verschiedene Einsatzszenarien entwickeln sollen. Bei dieser Aufgabe spielt eine unternehmensweite Einbettung der Systeme in eine homogene Softwarelandschaft keine Rolle. Aus diesem Grund sind die derzeitigen Methoden auch nicht dafür vorgesehen, von einem unternehmensweiten Geschäftsprozeß als Ausgangspunkt für eine Entwicklung auszugehen.

Dies alles führt zu einer Softwarelandschaft, in der Geschäftsprozesse nicht durchgängig, kaum wiederverwendbar und redundant auf mehrere Systeme verteilt sind. Mit einer solchen Softwarelandschaft sind die oben genannten Eigenschaften Kundenorientierung, Flexibilität und eine globale Ausrichtung nicht möglich. Durch die redundante, nicht durchgängige Abbildung von Geschäftsprozessen auf mehrere isolierte Systeme bedeutet die Änderung eines Prozesses die Änderung mehrerer Systeme. Und die verschiedenen Architekturen und Technologien der Systeme bewirken eine unterschiedliche Bindung der Funktionalität an Technik und Infrastruktur und dadurch wird eine nachträgliche Integration der einzelnen Systeme schwierig.

Um den Anforderungen gerecht zu werden und ein Business Process Reengineering erfolgreich abschließen zu können, sind offene, unternehmensweite Systemen notwendig. Offene, unternehmensweite Systeme müssen in einem geeigneten Prozeß entwickelt werden, der nicht nur das einzelne System, sondern auch die bestehenden, wiederverwendbaren Bausteine berücksichtigt. Die Abbildungen der Geschäftsprozesse in solchen Entwicklungsprozessen muß unternehmensweit in einem einheitlichen Rahmen erfolgen.

Ein solcher Rahmen ist das Ergebnis der vorliegenden Arbeit. Entwickelt wurde ein neuartiges methodisches Framework, das die Entwicklung offener, unternehmensweiter Systeme durch die Einführung eines grundlegenden Metamodells auf der Basis des Business Object Paradigma, darauf basierender, integrierter Beschreibungstechniken, eines unternehmensweiten Vorgehensmodells und eines umfassenden Werkzeugkonzeptes ermöglicht. Das methodische Framework umfaßt die Erarbeitung der folgenden Ergebnisse:

- Die Anforderungen des Business Process Reengineering auf den Entwurf und Entwicklung von Informationssystemen analysiert und gezeigt, daß die bisherigen, von Softwareunternehmen geprägten, projekt- und anwendungsfallbezogenen Methoden unzureichend sind für ein erfolgreiches

Business Process Reengineering.

- Offene, unternehmensweite Systeme werden als geeignetes Konzept vorgestellt, um unternehmensweite Geschäftsprozesse auf Informationssysteme abbilden zu können. Zur Entwicklung solcher Systeme wird ein Framework bestehend aus Metamodell, Beschreibungstechniken, Vorgehensmodell und Werkzeugkonzept vorgeschlagen.

- Auf Grundlage des in dieser Arbeit eingeführten Begriffes der Mehrfachnutzung wird das Konzept der Business Objects als grundlegendes Element für die Modellierung und Realisierung von offenen, unternehmensweiten Systemen propagiert. Ein umfassendes Metamodell definiert und verbindet die einzelnen Aspekte eines Business Object Modells.

- Erstmals wird ein Metamodell für Business Objects angegeben, das eine integrierte Modellierung von arbeitsteiligen Vorgängen ermöglicht und somit eine durchgängige und einheitliche Modellierung aller Elemente der Geschäftswelt ohne Brüche ermöglicht.

- Auf Basis des Metamodells wird ein integrierter Satz an Diagrammtypen als Profil der UML angegeben, der verschiedene Sichten auf alle Aspekte eines Business Object Modells ermöglicht.

- Ein eigenständiges Vorgehensmodell auf Grundlage des Unified Process beschreibt Rollen, Aktivitäten und Phasen eines Prozesses zur Entwicklung offener, unternehmensweiter Systeme auf Grundlage des Business Object Metamodells und der entwickelten Modellierungstechniken. Es berücksichtigt dabei speziell das von anderen Modellen nicht abgedeckte Anwendungsfall-übergreifende und Geschäftsprozeß-orientierte Vorgehen zur Entwicklung von Informationssystemen für *ein* Unternehmen.

- Basierend auf allen oben genannten Ergebnissen wird ein dreistufiges Werkzeugkonzept zur Modellierung eines Business Object Modells, zur Anreicherung des Modells um infrastrukturelle Komponenten und zur Abbildung des Modells auf eine Business Object Infrastruktur vorgestellt. Es umfaßt neben der XML-basierten eXtended Business Object Definition Language (XBODL) eine Abbildung aller Metamodellkonzepte auf die Enterprise Java Beans Infrastruktur.

Ein wesentlicher Beitrag der Arbeit ist das Business Object Metamodell, auf dem alle weiteren Framework-Bestandteile basieren. Es vereint die Konzepte bisheriger Metamodelle mit der Möglichkeit, arbeitsteilige Vorgänge auf Business Objects abzubilden. Der Aufbau des Metamodells erfolgte systematisch auf Basis

des MOF-Metametamodells und nach definierten Entwurfsgrundsätzen. Dies gewährleistet einen einheitlichen, strukturierten Zugriff auf die Eigenschaften der Modellelemente in einem MOF-konformen Werkzeug. Viele Eigenschaften des Metamodells wurden durch eindeutige Regeln in OCL definiert. Dies sichert die konsistente Instantiierung durch Business Object Modelle. Wichtig ist zudem, daß das Business Object Metamodell unabhängig von allen Middleware- oder Workflow-Infrastrukturen gehalten wurde. Dies gewährleistet ein Business Object Modell auf der Basis des Metamodells, das nicht schon beim nächsten Technologiewechsel veraltet ist und geändert werden muß.

Eine wichtige Leistung besteht in der Vorstellung von integrierten, UML-basierten Beschreibungstechniken. Sie bieten die Möglichkeit, verschiedene Sichten auf ein Business Object Modell zu erhalten. Dabei decken die entwickelten Diagrammarten alle Systemkonzepte des Business Object Metamodells ab und sind damit in der Lage, ein Business Object System vollständig zu modellieren. Alle Diagrammarten wurden auf das gemeinsame Business Object Metamodell abgebildet und die Bedeutung der Modellelemente auf struktureller Ebene festgelegt. Die Beschreibungstechniken werden durch ein Profil der UML definiert und können damit mit jedem UML-Werkzeug, das diese UML-Erweiterungsmöglichkeit unterstützt, bearbeitet werden.

Ein weiteres Ergebnis ist die Berücksichtigung der Besonderheiten einer Entwicklung von unternehmensweiten Anwendungen durch die Angabe eines Vorgehensmodells. Dieses neuartige Vorgehensmodell erweitert den Unified Process um diejenigen Workflows, Produkte und Akteure, die notwendig sind, ein Business Object System auf Basis des in dieser Arbeit vorgestellten Frameworks, das heißt dem Metamodell, der Beschreibungstechniken und des Werkzeugkonzeptes, zu entwickeln. Außerdem wurde ein angepaßtes Standard-Vorgehensmodell zur Entwicklung von Business Object Anwendungen definiert, das beide Entwicklungsprozesse – der des Business Object Systems und der der Business Object Anwendung – miteinander verzahnt. Die entsprechenden Anknüpfungspunkte des Standard-Vorgehensmodells RUP an das unternehmensweite Vorgehen wurden hinzugefügt.

Ohne eine integrierte Werkzeugunterstützung zur Sicherstellung eines korrekten und konsistenten Business Object Modells und seiner automatischen Generierung und Abbildung auf eine Infrastruktur wäre ein unternehmensweites Vorgehen auf der Basis von Business Objects nur schwer durchführbar.

Abschließend wird das Framework ergänzt durch ein integriertes Werkzeugkonzept. Es umfaßt die Konzeption eines UML-basierten Modellierungswerkzeugs auf der Basis des kommerziellen UML-Tools Together zur Darstellung und Be-

arbeitung der verschiedenen Sichten auf das Business Object Modells. Weiterhin wird eine Sprache zur Beschreibung eines werkzeugunabhängigen Infrastrukturmodells vorgestellt: die eXtendet Business Object Description Language XBODL auf der Basis von UML. Die Zwischendarstellung durch die XBODL ermöglicht die Weiterbearbeitung des Business Object Modells zur deklarativen Erweiterung des Modells um technische Aspekte. Das Werkzeugkonzept wird vervollständigt durch einen Vorschlag zur Abbildung des Infrastrukturmodells auf eine geeignete Infrastruktur, dem Enterprise JavaBeans-Framework. Die Tragfähigkeit des Werkzeugkonzepts wurde durch eine Realisierung einer im Rahmen einer Diplomarbeit [Frö01] entstandenen prototypischen Werkzeugumgebung gezeigt.

Generell wurde großen Wert auf eine Erweiterung der Framework-Elemente und des Frameworks an sich geachtet. So ist die Erweiterung des Metamodells um Elemente zur Abfrage und Manipulation von Metainformationen denkbar. Durch sogenannte Reflexion-Techniken können zur Laufzeit Komponenten des Business Object Systems geändert oder hinzugefügt werden. Sinnvoller Einsatz einer solchen Technik ist die Ad-hoc-Änderung eines Business Object Workflows durch Ändern, Löschen oder Hinzufügen von Action- und Flow-Elementen.

Auch die Modellierung könnte noch durch zusätzliche Sichten unterstützt werden. So ist eine Protokolldarstellung von Business Object Interaktionen durch Message-Sequence Charts (MSCs) – in der UML Sequenz-Diagramme genannt – denkbar. Wenn die Elemente der MSCs auf das Business Object Metamodell abgebildet werden, ist auf der Basis der definierten Protokolle eine Sicherstellung eines gewünschten Ablaufs zur Laufzeit des Business Object Systems möglich.

Anstelle der jetzigen Lösung, die Methoden nur in Form von Java-Anweisungen zu spezifizieren, ist der Einsatz von Zustandsautomaten möglich. Zustandsautomaten hätten gegenüber programmiersprachlichen Lösungen den Vorteil, das Verhalten von Business Objects durch Modellchecker auf Modellebene überprüfen zu können, ohne das Business Object Modell zuerst auf eine Infrastruktur abbilden zu müssen. Der Nachteil dabei ist, daß für die Zustandsautomaten das Metamodell erweitert werden müßte.

Eine Erweiterung der Werkzeugumgebung wären Simulatoren für das Business Object Modell. Der Weg dahin ist nicht weit, denn es müßte nur das Backend des Generators, der aus dem Infrastrukturmodell das eigentliche Business Object System generiert, ausgetauscht werden. Mit einem Simulator-Backend wäre es dann möglich, auf der Basis des Business Object Modells bestimmte Testfälle oder Prozeßabläufe durchzuführen. Der Vorteil hiervon wäre ein verkürzter Entwicklungszyklus.

Und schließlich wäre eine Ausdehnung des vorgestellten, methodischen Frameworks auf ein unternehmensübergreifendes Vorgehen möglich. Bei der jetzigen, unternehmensweiten Konzeption spielt die Wiederverwendung von Business Objects zur Entwicklungszeit keine Rolle. Ein unternehmensübergreifendes Vorgehen würde eine Lösung der bis heute problematischen Wiederverwendung von fachlichen Komponenten darstellen. Die Hauptschwierigkeit einer solchen Ausdehnung liegt aber nicht in der Erweiterung des Frameworks an sich, sondern in der Frage, ob die Geschäftswelt zweier Unternehmen sich so gleicht, daß die gleiche Business Object Architektur vorliegt. Die für die Beantwortung der Frage notwendigen Techniken zur vollständigen, normalisierten Verhaltensbeschreibung von Komponenten sind zur Zeit noch nicht im größeren Stil einsetzbar.

Literaturverzeichnis

[AF98] Paul Allen and Stuart Frost. *Component-Based Development for Enterprise Systems*. Cambridge University Press, New York, 1998.

[AG90] C. Ashworth and M. Goodland. *SSADM – A Practical Approach*. McGraw Hill, 1990.

[Arn91] R. S. Arnold. Risks of reengineering. In *Proc. Reverse Eng. Forum*, St. Louis, April 1991.

[BD92] A. P. Bröhl and W. Dröschel, editors. *Das V-Modell in der Anwendungsentwicklung – Standard und Leitfaden*. Oldenbourg, München, 1992.

[BJNR98] Bohrer, Johnson, Nilson, and Rubin. Business Process Components for Distributed Object Applications. *Communications of the ACM 41, Nr. 6*, pages 43–48, 1998.

[Boe88] B. W. Boehm. A Spiral Model of Software Development an Enhancement. *IEEE Computer*, 25(12), 1988.

[Boo91] Grady Booch. *Object Oriented Design with Applications*. The Benjamin/Cummings Publishing Company, Inc., 1991.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modelling Language (UML) Users Guide*. Addison Wesley, 1999.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems – FOCUS on Streams, Interfaces and Refinement*. Springer, 2001.

[BSK96] Israel Ben-Shaul and Gail Kaiser. *A Paradigm for Decentralized Process Modeling*. Kluwer Academic, Boston, 1996.

[Buc99] Alejandro Buchmann, editor. *Datenbanksysteme in Büro, Technik und Wissenschaft: 8. GI-Fachtagung 1.-3. März*. Springer, Berlin, 1999.

[CD94] S. Cook and J. Daniels. *Designing Object Systems: Object-Oriented Modelling with Syntropy*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[CN91] Brad Cox and Andrew Novobilski. *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, New York, 1991.

[CN98] Wolfram Conen and Gustaf Neumann, editors. *Coordination Technology for Collaborative Applications: Organizations, Processes and Agents*, Lecture Notes in Computer Science 1364. Springer, Berlin, 1998.

[Cox96] Brad Cox. *Superdistribution: Objects as Property on the Electronic Frontier*. Addison-Wesley, New York, 1996.

[DW99] Desmond F. D'Souza and Alan C. Wills. *Components and Frameworks with UML: The Catalysis Approach.* Addison-Wesley, New York, 1999.

[Ern98] J. Ernst. Cdif – xml-based transfer format, 1998.

[ES98] Peter Eeles and Oliver Sims. *Building Business Objects*. John Wiley & Sons, 1998.

[FR99] Robert France and Bernhard Rumpe, editors. *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723. Springer, 1999.

[Frö01] Markus Fröhler. Konzeption und prototypische Werkzeugunterstützung einer Methode zur Modellierung unternehmensweiter Daten und Prozesse basierend auf der UML. Master's thesis, Institut für Informatik, Technische Universität München, 2001.

[Fuj00] i-Flow: Empowering Enterprises to Manage Business Process Change. Fujitsu Software Corporation, www.i-flow.com, 2000.

[GAO95] David Garlan, Robert Allen, and John Ockerbloom. Architectural mismatch: Why reuse is so hard. *IEEE Software*, 12, 1995.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Reading, Massachusetts, 1994. ISBN: 0-201-63361-2.

[GHSY97] Ian Graham, Brian Henderson-Sellers, and Houman Younessi. *The OPEN Process Specification*. Addison-Wesley, New York, 1997.

[Gos98] Sanjiv Gossain. *Object Modelling and Design Strategies*. Cambridge University Press, 1998.

[GTP] Software architectures. Dagstuhl-Seminar-Report 106, 20.02.-24.02.95.

[GY96] James Gosling and Frank Yellin. *The Java Application Programming Interface, Vol. 2*. Addison Wesley, 1996.

[HC93] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution1*. Harper Business, New York, 1993.

[Hir96] Robert Hirschfeld. Three-Tier Distribution Architecture. In *PLoP '96 Proceedings, Monticello IL*, 1996.

[HMPS98] Wiebe Hordijk, Sascha Molterer, Barbara Paech, and Chris Salzmann. Working with business objects - a case study. In D. Patel, J. Sutherland, and J. Miller, editors, *Business Object Design and Implementation II*. Springer Verlag, 1998.

[HMS99] Wiebe Hordijk, Sascha Molterer, and Chris Salzmann. The reuse benefit of business objects: A controlled experiment. 1999.

[HS99] Peter Herzum and Oliver Sims. *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. John Wiley & Sons, New York, 1999.

[HSSS96] Franz Huber, Bernhard Schätz, Alexander Schmidt, and Katharina Spies. Autofocus – a tool for distributed system specification. In *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96)*, number 1135 in LNCS. Springer Verlag, 1996.

[IBM93] Open Blueprint Introduction. Technical Report, IBM Corporation, Dezember 1993.

[Jab95] Stefan Jablonski. *Workflow-Management-Systeme*. IVP Verlag, 1995.

[JBR98] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, New York, 1998.

[JBS97] Stefan Jablonski, Markus Boehm, and Wolfgang Schulze, editors. *Workflow-Management: Entwicklung von Anwendungen und Systemen. Facetten einer neuen Technologie*. dpunkt.verlag, Heidelberg, 1997.

[JCJO92] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering (OOSE): A Use Case Driven Approach*. Addison-Wesley, Reading, Massachusetts, 1992.

[JGJ97] Ivar Jacobson, Martin Griss, and Patrik Johnson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison Wesley, 1997.

[JO99] Matthias Jarke and Andreas Oberweis, editors. *Advanced Information Systems Engineering, 11th International Conference, CAiSE'99, Heidelberg,* Lecture Notes in Computer Science 1626. Springer, Berlin, 1999.

[Kau94] Achim Kaufmann. *Software-Reengineering: Analyse, Restrukturierung und Reverse-Engineering von Anwendungssystemen*. Oldenburg, München, 1994.

[KCD99] Pinar Koksal, Ibrahim Cingil, and Asuman Dogac. A component-based workflow system with dynamic modifications. In Pinter and Tsur [PT99].

[KFN99] Cem Kaner, Jack Falk, and Hung Q. Nguyen. *Testing Computer Software*. Wiley & Sons, New York, 1999.

[KNS92] G. Keller, M. Nüttgens, and A. W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, University of Saarland, Saarbrücken, 1992.

[Kor01] Axel Korthaus. *Komponentenbasierte Entwicklung computergestützter betrieblicher Informationssysteme*. PhD thesis, Fakultät für Betriebswirtschaftslehre, Universität Mannheim, 2001.

[KP98] Kwang-Hoon Kim and Su-Ki Pak. Practical experiences and requirements on workflow. In Conen and Neumann [CN98].

[KR98] Haim Kilov and Bernhard Rumpe, editors. *Proceedings Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications)*. Institut für Informatik, Technische Universität München, 1998. Technical Report, TUM-I9813.

[KRS98] Haim Kilov, Bernhard Rumpe, and Ian Simmonds, editors. *Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, 1998. Technical Report, TUM-I9820.

[Kru92] C. W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131–183, 1992.

[Kru95] Philippe Kruchten. The 4+1 View of Architecture. *IEEE Software*, 12(6):45–50, 1995.

[Kru98] Phillipe Kruchten. *The Rational Unified Process (RUP)*. Addison-Wesley, New York, 1998.

[KS97] D. Kochut and M. Sheth. Orbwork: A reliable distributed corba-based workflow enactment systems meteor. Technical Report Technical Report UGA-CS-TR-97-001, University Georgia, Department Computer Science, 1997.

[Leh99] Franz Lehner. Software Reengineering vs. Business Reengineering – Konsequenzen für ein unternehmensweites Reengineering-Konzept. In J. Ebert and F. Lehner, editors, *Workshop Software-Reengineering, Bad Honnef 1999*, number 7-99 in Fachberichte Informatik, pages 87–94. Universität Koblenz-Landau, Institut für Informatik, September 1999.

[LM98] Winfried Lamersdorf and Michael Merz, editors. *Trends in Distributed Systems for Electronic Commerce (TREC'98). International IFIP/GI Working Conference, Hamburg, Juni 1998*, Lecture Notes in Computer Science 1402. Springer, Berlin, 1998.

[LR00] Frank Leymann and Dieter Roller. *Production Workflow: Concept and Techniques*. Prentice Hall, New York, 2000.

[LV99] Pierre Ladet and Francoise Vernadat, editors. *Integrated Manufacturing Systems Engineering*. Chapman and Hill, London, 1999.

[Mar00] Chris Marshall. *Enterprise Modelling with UML*. Addison-Wesley, 2000.

[Meg98] David Megginson. *Structuring XML Documents* . Prentice Hall, New York, 1998.

[Mit97] Christian Mittasch. Workflow-Management with BPAFrame. In W. Abramowicz, editor, *Procceedings of the International Conference of Business Information Systems'97 (BIS'97), Poznan, Polen*, 1997.

[MJ99] Dragoş-Anton Manolescu and Ralph E. Johnson. A Micro Workflow Framework for Compositional Object-Oriented Software Development. OOPSLA'99 Workshop on the Implementation and Application of Object-Oriented Workflow Management Systems, 1999.

[MN88] Pattie Maes and Daniele Nardi, editors. *Meta–Level Architecture and Reflection*. North Holland, 1988.

[MSKW96] John A. Miller, Amit P. Sheth, Krys J. Kochut, and Xuzhong Wang. CORBA-Based Run-Time Architectures for Workflow Management Systems. *Journal of Database Management, Special Issue on Multidatabases*, 7(1), 1996.

[OHE96] Orfali, Harkey, and Edwards. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, 1996.

[OMG92] Object Management Architecture Guide, Revision 2.0. Object Management Group, OMG document TC/92.11.1, 1992.

[OMG94] Business Application Architecture, Revision 2. Object Management Group, OMG document bom/95-04-01, 1994.

[OMG95] Common Facilities RFP-4 (Common Business Objects and Business Objects Facility). Object Management Group, OMG document bom/96-01-04, 1995.

[OMG96] Object Analysis and Design PTF - RFP 1. Object Management Group, OMG document ad/96-05-01, 1996.

[OMG97a] Workflow-Management Facility RFP. Object Management Group, OMG document cf/97-05-06, 1997.

[OMG97b] The Meta Object Facility (MOF) Specification – Version 1.1. Object Management Group, OMG document ad/97-10-02, 1997.

[OMG98a] Business Object Component Architecture (BOCA), Revision 1.2. Object Management Group, OMG document bom/98-05-03, 1998. OMG Business Object Domain Task Force: BODTF-RFP 1 Submission: Combined Business Object Facility.

[OMG98b] CORBAcomponents Specification 1.2. Object Management Group, OMG document cf/98-01-07, 1998.

[OMG98c] jointFlow Workflow Definition. Object Management Group, OMG document bom/98-01-28, 1998.

[OMG98d] Managed Object Facility. Object Management Group, OMG document cf/98-01-07, 1998.

[OMG99a] The Unified Modeling Language (UML) Specification – Version 1.3. Object Management Group, OMG document ad/99-06-08, 1999.

[OMG99b] The Meta Object Facility (MOF) Specification – Version 1.3. Object Management Group, OMG document ad/99-06-05, 1999.

[OMG99c] XML Metadata Interchange (XMI) Specification – Version 1.1. Object Management Group, OMG document ad/99-10-02, 1999.

[PJWJ98] Mike Papazoglou, Manfred Jeusfeld, Hans Weigand, and Matthias Jarke. Distributed, interoperable workflow support for electronic commerce. In Lamersdorf and Merz [LM98].

[Por98] Michael Porter. *Competitive Advantage: Creating and Sustaining Superior Performance (New edition)*. Simon & Schuster, 1998.

[Pri96] Robert Prins. *Developing Business Objects: A Framework-driven Approach*. McGraw-Hill, 1996.

[PRW96] Arnold Picot, Ralf Reichwald, and Rolf T. Wigand. *Die grenzenlose Unternehmung – Information, Organisation und Management; Lehrbuch zur Unternehmensführung im Informationszeitalter*. Gabler Verlag, Wiesbaden, 1996.

[PSM98] Patel, Sutherland, and Miller, editors. *Business Object Design and Implementation, Proceedings of the OOPSLA'96/'97/'98 Workshops*. Springer Verlag, London, 1998.

[PSM99] Patel, Sutherland, and Miller, editors. *Business Object Design and Implementation, Proceedings of the OOPSLA'99 Workshop, Denver*. Springer Verlag, London, 1999.

[PT99] Ron Pinter and Shalom Tsur, editors. *Next Generation Information Technologies and Systems (NGITS'99). 4th International Workshop, Zikhron-Yaakov, Israel, July 1999*, Lecture Notes in Computer Science 1649. Springer, Berlin, 1999.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design*. Prentice Hall, 1991.

[RJB98] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language (UML) Reference Manual*. Addison-Wesley, New York, 1998.

[RKB95] B. Rumpe, C. Klein, and M. Broy. Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme - SYSLAB Systemmodell. Technical Report TUM-I9510, Technische Universität München, 1995.

[RMS+99] R. Reichwald, K. Möslein, H. Sachenbacher, H. Englberger, and S. Oldenburg. *Telekooperation: Verteilte Arbeits- und Organisationsformen*. Springer, Berlin, 1999.

[Roh97] Michael Rohloff. An object oriented approach to business modelling. In Scholz-Reiter and Stickel [SRS97].

[Rom99] Ed Roman. *Mastering Enterprise JavaBeans and the Java 2 Platform Enterprise Edition*. Wiley & Sons, New York, 1999.

[RR00] J. Robbins and D. Redmiles. Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML. *Journal on Information and Software Technology*, 42(12):79–89, 2000.

[Rum98] Bernhard Rumpe. A Note on Semantics (with an Emphasis on UML). In Kilov and Rumpe [KR98]. Technical Report, TUM-I9813.

[Sch92] August-Wilhelm Scheer. *Architecture of Integrated Information Systems: Foundations of Enterprise Modelling*. Springer-Verlag, Berlin, 1992.

[Sch96] Thomas Schäl. *Workflow Management Systems for Process Organisations*. Lecture Notes in Computer Science 1096. Springer, Berlin, 1996.

[Sch97] Wolfgang Schulze. Evaluation of the Submissions to the Workflow Management Facility RFP. OMG document bom/97-09-02, 1997.

[Sch99] Wolfgang Schulze. *Workflow-Management für CORBA-basierte Anwendungen*. PhD thesis, Technische Universität Dresden, 1999.

[SH99] A.-W. Scheer and M. Hoffmann. From business process model to application systems – developing an information system with the house of business engineering (hobe). In Jarke and Oberweis [JO99].

[She97] Robert Shelton. Business Objects - Definition, Taxonomie and Abstraction, Presentation Slides of the OMG Meeting in Stresa. zitiert in [Sch99], mar 1997.

[Sim94] Oliver Sims. *Business Objects: Delivering Cooperative Objects for Client-Server*. McGraw-Hill, 1994.

[SK98] Martin Schader and Axel Korthaus, editors. *The Unified Modeling Language (UML): Technical Aspects and Applications*. Physica, Heidelberg, 1998.

[SLJR94] Kathy Spurr, Paul Layzell, Leslie Jennison, and Neil Richards, editors. *Business Objects Software Solutions*. John Wiley Sons Ltd., Chichester, England, 1994. ISBN: 0-471-95187-0.

[SLPFE98] Monika Saksena, Maria Larrondo-Petrie, Robert France, and Matthew Evett. Extending Aggregation Constructs in UML. In Jean Bézivin and Pierre-Alain Muller, editors, *The Unified Modeling Language, UML'98 - Beyond the Notation. First International Workshop, Mulhouse, France, June 1998, Selected Papers*, volume Lecture Notes on Computer Science 1618, pages 434–441. Springer, Berlin, 1998.

[SM88] S. Shlaer and S. Mellor. *Object-Oriented Systems Analysis*. Prentice-Hall, 1988.

[Sok95] P. Sokol. From edi to electronic commerce – a business initiative, 1995.

[SPC+97] Sutherland, Patel, Casanave, Hollowell, and Miller, editors. *Business Object Design and Implementation, Proceedings of the OOPSLA'95 Workshop, Austin*. Springer Verlag, London, 1997.

[SRS96] Bernd Scholz-Reiter and Eberhard Stickel, editors. *Business Process Modelling*. Springer-Verlag, Berlin, 1996.

[SRS97] Bernd Scholz-Reiter and Eberhard Stickel, editors. *Business Process Modelling*. Springer, Berlin, 1997.

[SSSM99] Stefan Sarstedt, Günter Sauter, Jürgen Sellentin, and Bernhard Mitschang. Integrationskonzepte für heterogene anwendungssysteme bei daimler-chrysler auf basis internationaler standards. In Buchmann [Buc99].

[SUN96a] The Java Language Specification 1.0. SUN Microsystems Inc., 1996.

[SUN96b] The Java Virtual Machine Specification. SUN Microsystems Inc., 1996.

[SUN99a] Enterprise JavaBeans Specification 1.1. SUN Microsystems Inc., http://java.sun.com, 1999.

[SUN99b] Java 2 Platform Enterprise Edition Specification, Version 1.2. SUN Microsystems Inc., http://java.sun.com, 12 1999.

[SUN00] Enterprise Java Beans Specification 2.0. Sun Microsystems Inc., http://java.sun.com, 5 2000.

[Tay94] David A. Taylor. *Business Engineering with Object Technology*. John Wiley Sons, Inc., 1994. ISBN: 0-471-04521-7.

[TM97] SAP AG Technology Marketing. R/3 system: Benefits of the business framework, 1997.

[Tog01] Together ControlCenter 5.5 API Specification. TogetherSoft Corp., www.togethersoft.com, 2001.

[vdHPJ99] Willem-Jan van den Heuvel, Mike Papazoglou, and Manfred A. Jeusfeld. Configuring business objects from legacy systems. In Jarke and Oberweis [JO99].

[Ver00] The Verve Workflow Management System. Verve Inc., www.verve.com, 2000.

[Wes99] Mathias Weske. Workflow management through distributed and persistent corba workflow objects. In Jarke and Oberweis [JO99].

[WFM94] The Workflow Reference Model, Workflow Management Coalition. WFMC-TC-1003, Version 1.1, 1994.

[WK99] Jos Warmer and Anneke Kleppe. OCL: The Constraint Language of the UML. *Journal of Object-Oriented Programming*, 12(1):10–28, 1999.

[WVBMP98] Weske, Vossen, Bauzer-Medeiros, and Pires. Workflow Management in Geoprocessing Applications. Technical Report Fachbericht Angewandte Mathematik und Informatik 04/98-I, Universität Münster, 1998.

[WWWKD97] D. Wodtke, J. Weissenfels, G. Weikum, and A. Kotz-Dittrich. The mentor project: Steps towards enterprise-wide workflow management. In *Proceedings of the IEEE International Conference on Data Engineering*, 1997.

[Zah00] Ron Zahavi. *Enterprise Application Integration with CORBA: Components and Web-Based Solutions*. John Wiley, New York, 2000.

[Zuk99] Olaf Zukunft. Waterloo-m: Ein datenbankbasiertes workflow-managementsystem für mobile benutzer. In Buchmann [Buc99].

Anhang A

Metamodell: Zusammenfassung

Das Business Object Metamodell ist in drei Teile (Packages) aufgeteilt: Allgemeine Modellelemente (“Base”), Modellelemente zur Beschreibung von strukturellen Eigenschaften (“Structure”) und Modellelemente zur Beschreibung von dynamischen Eigenschaften (“Dynamics”).

Die allgemeinen – und allesamt abstrakten – Modellelemente des Packages “Base” bestimmen gemeinsame Eigenschaften des gesamten Business Object Metamodells: Klassifizierung, Generalisierung, Namensräume, Typen und die konkreten strukturellen Eigenschaften.

Das Package lehnt sich größtenteils an die Grundstruktur des MOF-Metamodells der OMG [OMG99b] an. Dies gewährleistet einen einheitlichen, strukturierten Zugriff auf die Eigenschaften der Modellelemente in einem MOF-konformen Werkzeug. Die folgende Tabelle faßt die Elemente des Packages und ihre Beschreibung zusammen:

Element	**Beschreibung**
ModelElement	Oberste Klasse im Metamodell. Alle Modellelemente des Modells erben dessen Attribut Name und die Referenz auf einen Namensraum.
Namespace	Basisklasse für alle Elemente des Metamodells, die andere Modellelemente in ihrem Namensraum enthalten können. Das Namensraumkonzept wird für unterschiedliche Eigenschaften im Metamodell verwendet.

Generalizable-Element	Von dieser Klasse erben alle Modellelemente, die mit sich selbst innerhalb einer Vererbungsrelation verknüpft werden können.
TypedElement	Element zur Abstrahierung von Modellelementen, die einen Typ als Teil ihrer Definition benötigen.
Classifier	Basisklasse aller Modellelemente, die sich durch bestimmte strukturelle oder dynamische Eigenschaften (Features) klassifizieren lassen.
Feature	Zum Hinzufügen von Eigenschaften zu einem Modellelement. Diese Eigenschaften können auf Ebene M1 den Modellklassen oder den einzelnen Instanzen dieser Modellelemente zugewiesen werden.
StructuralFeature	Basisklasse für alle strukturellen Eigenschaften von Modellelementen.
DynamicFeature	Basisklasse für alle dynamischen Eigenschaften von Modellelementen.

Aufbauend auf die allgemeinen Modellelemente werden zwei Packages (“Structure” und “Dynamics”) eingeführt die sich mit den Modellelementen befassen, die zur Modellierung von Business Object Modellen dienen. Diese Packages sind originär für diese Arbeit entwickelt worden und entsprechen nicht den traditionellen Metamodellen zur Entwicklung objektorientierten Systemen, wie zum Beispiel der UML.

Zunächst werden in folgender Tabelle die Modellelemente des Packages “Structure” zusammengefaßt. Sie dienen der Klassifizierung und Strukturierung von fachlichen Konzepten der Geschäftswelt in einem Business Object Modell.

Element	**Beschreibung**
Proxy	Blendet Modellelemente in andere Domains ein. Ein Proxy ist dabei keine Kopie, sondern – wie der Name schon verdeutlicht – ein Stellvertreter des eigentlichen Modellelements.
BusinessObject (BO)	Ein BO ist die Generalisierung für alle Konzepte der Geschäftswelt, die mit einem Business Object Modell modelliert werden sollen.

BusinessEntity-Object (BEO)	Modellelement zur Abbildung eines passiven, zustandsbehafteten Konzeptes der Geschäftswelt.
BusinessService-Object (BSO)	Ein BSO ist ein Element, das transaktionale Manipulationen auf ein oder mehrere BOs modelliert.
BusinessWorkflow-Object (BWO)	Abstraktion für alle in das Modell abzubildende arbeitsteiligen Vorgänge (Workflows). Kann selbst wieder BWOs als Subworkflows enthalten.
BusinessTaskObject (BTO)	Modellelement zur Modellierung aller Arbeitsaufgaben der Geschäftswelt. Arbeitsaufgaben spielen als kleinste Teilschritte in einem BWO eine Rolle und sind dediziert Akteuren (BAOs) zugewiesen.
BusinessActor-Object (BAO)	Zur generischen Modellierung aller aktiven Elemente der Geschäftswelt. Dies sind in der Regel die Personen eines Unternehmens, es können jedoch auch Softwaresysteme mit BAOs gekapselt werden.
Attribute	Modellelement zur Beschreibung von Attributen als strukturelle Eigenschaft von Klassen (Classifier) im Modell.
Association	Basiselement zur Modellierung von Beziehungen bzw. Abhängigkeiten zwischen BOs.
AssociationEnd	Hilfselement zur Bestimmung der Eigenschaften des Association-Elements. Für jeden Partner in einer Association gibt es ein AssociationEnd.

Neben dem Package “Structure” mit strukturellen Eigenschaften existiert im Metamodell noch ein Package “Dynamics” mit denjenigen Modellelementen, die im Business Object Modell mögliche Veränderungen bzw. Manipulationen der Struktur darstellen. Dies umfaßt Manipulationen auf Business Objects, Benachrichtigungen von Business Objects durch Events und die Kontrollflußeigenschaften von arbeitsteiligen Vorgängen. In nachfolgender Tabelle werden die Modellelemente des Packages zusammengefaßt:

Element	**Beschreibung**
Operation	Zur Modellierung von Schnittstellen der Manipulationen (Lesen, Ändern) auf den jeweiligen Zustand (Attribute) ein oder mehrerer BOs.

Method	Realisierung der durch Operation deklarierten Manipulationen. In dieser Arbeit beschreibbar durch ein Subset von Java.
Parameter	Zur Parametrisierung von Operationen und Events.
Event	Modellelement zur Beschreibung von Events und deren Produzenten und Konsumenten im Business Object System.
Flow	Zur Definition eines Kontrollflusses innerhalb eines BWOs.
Action	Hilfselemente zur Definition eines Kontrollflusses innerhalb eines BWOs (Start- und Endpunkt, bedingte Verzweigung und parallele Verzweigung).

Anhang B

Metamodell: Regeln

```
------------------------------------------------------
-- Business Object Metamodel Rules     bomm-rules.ocl -
--                                                    -
-- Version: 1.0 (00-12-19)                            -
-- Author:  Sascha Molterer                           -
--          molterer@in.tum.de                        -
------------------------------------------------------

context ModelElement
inv:
  not self.oclIsTypeOf(Domain) implies
    self.namespace -> size = 1

context Namespace
inv:
  self.contents.forAll(e1, e2 |
    e1.name = e2.name implies e1 = e2)

context GeneralizableElement
inv:
  self.allSupertypes() -> forAll(s | s <> self)

context GeneralizableElement
inv:
  self.supertypes ->
       forAll(s | s.oclType() = self.oclType())
```

```
context GeneralizableElement
inv:
  let superContents = self.allSupertypes() ->
    collect(s | s.contents) in
  self.contents ->
    forAll(n1 | superContents ->
      forAll(n2 | n1.name = n2.name implies m1 = m2))

context Domain
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(Domain) or
    e.oclIsKindOf(BusinessEntityObject) or
    e.oclIsKindOf(BusinessServiceObject) or
    e.oclIsKindOf(BusinessWorkflowObject) or
    e.oclIsKindOf(BusinessSubject) or
    e.oclIsKindOf(DataType) or
    e.oclIsKindOf(Association) or
    e.oclIsKindOf(Event) or
    e.oclIsKindOf(Proxy) )

context Domain
inv:
  self.supertypes -> isEmpty and
  self.isRoot and
  self.isLeaf and
  not self.isAbstract

context Proxy
inv:
  self.namespace <>  self.master -> namespace

context BusinessEntityObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(DataType} or
    e.oclIsKindOf(Reference) or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )
```

```
context BusinessServiceObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(DataType} or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )

context Association
inv:
  self.contents.forAll( e |
                       e.oclIsKindOf(AssociationEnd) )

context Association
inv:
  self.supertypes -> isEmpty and
  self.isRoot and
  self.isLeaf and
  not self.isAbstract

context Association
inv:
  self.contents ->
    select(a | a.oclIsTypeOf(AssociationEnd) )
      -> size = 2

context AssociationEnd
inv:
  self.type.oclIsTypeOf(BusinessObject)

context AssociationEnd
inv:
  self.aggregation <> #independent
    implies (self.container.contents ->
          select(a | a.oclIsKindOf(AssociationEnd)
                     and a <> self)) = #independent
    select(a | a.oclIsTypeOf(AssociationEnd) )
                                        -> size = 2

context Operation
```

```
inv:
  self.contents.forAll( e | e.oclIsKindOf(Parameter} )

context BusinessWorkflowObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(BusinessWorkflowObject) or
    e.oclIsKindOf(BusinessTaskObjects) or
    e.oclIsKindOf(BusinessActorObjects) or
    e.oclIsKindOf(Flow) or
    e.oclIsKindOf(Action) or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )

context BusinessWorkflowObject
inv:
  self.supertypes -> isEmpty and
  self.isRoot and
  self.isLeaf and
  not self.isAbstract

context BusinessTaskObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(DataType) or
    e.oclIsKindOf(PropertyAssociation) or
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )

context BusinessActorObject
inv:
  self.contents.forAll( e |
    e.oclIsKindOf(Operation) or
    e.oclIsKindOf(Attribute) or
    e.oclIsKindOf(Event) )

context Flow
inv:
  (self.fromClassifier.
```

```
     oclIsKindOf(BusinessActivityObject) or
self.fromClassifier.
     oclIsKindOf(BusinessWorkflowObject) or
self.fromClassifier.
     oclIsKindOf(Action)) and
(self.toClassifier.
     oclIsKindOf(BusinessActivityObject) or
self.toClassifier.
     oclIsKindOf(BusinessWorkflowObject) or
self.toClassifier.
     oclIsKindOf(Action))
```

Anhang C

XBODL Definition

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- eXtended Business Objects Definition Language    -->
<!--                  (XBODL)                         -->

<!-- XBODL version: 1.2 (01-03-15)                    -->
<!-- XBODL author: Sascha Molterer                    -->
<!--               Institut fuer Informatik           -->
<!--               Technische Universitaet Muenchen   -->
<!--               molterer@cs.tum.edu                -->

<!--
All valid xbodl files must contain the following
declaration:

<!DOCTYPE xbodl PUBLIC "-//Institut fuer Informatik//
DTD eXtended Business Object Definition Language//EN"
"http://www4.cs.tum.edu/~molterer/xbodl/xbodl_1_2.dtd">
-->

<!--
root section of model ----------------------------------
-->
<!ELEMENT xbomodel (elements,references,extensions+)>

<!--
```

```
three main sections: elements, references, extensions ---
-->
<!ELEMENT elements (beo*,bso*,bwo*,bto*,bao*,
                    domain+,event*,datatype*)>
<!ELEMENT references (flow*,consumes*,produces*,
                      association*,generalization*)>
<!ELEMENT extensions (idxdef*,typexdef*,xdef)>
<!ATTLIST extensions
  platform CDATA #IMPLIED
>


<!--
definitions for elements section ------------------------
-->
<!ELEMENT domain EMPTY>
<!ATTLIST domain
  id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT beo (attribute*,operation*)>
<!ATTLIST beo
  id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT bso (attribute*,operation*)>
<!ATTLIST bso
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  isSingleton (true|false) #IMPLIED
>
<!ELEMENT bwo (attribute*,operation*,action+)>
<!ATTLIST bwo
  id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT bto (operation*,attribute*)>
<!ATTLIST bto
  id CDATA #REQUIRED
  name CDATA #REQUIRED
>
```

```
<!ELEMENT action EMPTY>
<!ATTLIST action
  id CDATA #REQUIRED
  kind (start|stop|if|fork|join) #REQUIRED
>
<!ELEMENT bao (attribute*,operation*)>
<!ATTLIST bao
  id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT event (parameter+,method?)>
<!ATTLIST event
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  scope (class|instance) #IMPLIED
>
<!ELEMENT datatype EMPTY>
<!ATTLIST datatype
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  typeCode CDATA #REQUIRED
>
<!ELEMENT attribute EMPTY>
<!ATTLIST attribute
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  multiplicity CDATA #IMPLIED
  isChangable (true|false) #IMPLIED
  typeRef CDATA #REQUIRED
>
<!ELEMENT operation (method?,parameter+)>
<!ATTLIST operation
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  isAbstract CDATA #IMPLIED
  isQuery (true|false) #IMPLIED
>
<!ELEMENT method (#PCDATA)*>
<!ATTLIST method
  id CDATA #REQUIRED
>
```

```
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  direction (in|out|inout|return) #REQUIRED
  defaultValue CDATA #IMPLIED
  typeRef CDATA #REQUIRED
>

<!--
definitions for references section ----------------------
-->
<!ELEMENT association (associationend+)>
<!ATTLIST association
  id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT associationend EMPTY>
<!ATTLIST associationend
  id CDATA #REQUIRED
  name CDATA #REQUIRED
  multiplicity CDATA #IMPLIED
  aggregation (independent|aggregate|composite) #IMPLIED
  elementRef CDATA #REQUIRED
>
<!ELEMENT generalization EMPTY>
<!ATTLIST generalization
  id CDATA #REQUIRED
  elementRef CDATA #REQUIRED
  supertypeRef CDATA #REQUIRED
>
<!ELEMENT flow EMPTY>
<!ATTLIST flow
  id CDATA #REQUIRED
  fromRef CDATA #REQUIRED
  toRef CDATA #REQUIRED
>
<!ELEMENT consumes EMPTY>
<!ATTLIST consumes
  id CDATA #REQUIRED
  consumerRef CDATA #REQUIRED
```

```
  eventRef CDATA #REQUIRED
>
<!ELEMENT produces EMPTY>
<!ATTLIST produces
  id CDATA #REQUIRED
  producerRef CDATA #REQUIRED
  eventRef CDATA #REQUIRED
>

<!--
definitions for extensions section ----------------------
-->
<!ELEMENT idxdef ANY>
<!ATTLIST idxdef
  targetRef CDATA #REQUIRED
>
<!ELEMENT typexdef ANY>
<!ATTLIST typexdef
  target (domain|beo|bso|bwo|bto|bao|event|datatype|
          parameter|method|operation|attribute|
          action|generalization|flow|consumes|
          produces|association|associationend) #REQUIRED
>
<!ELEMENT xdef ANY>

<!-- end of xbodl definition -->
```