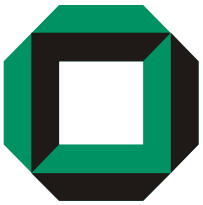
Universität Karlsruhe – Fakultät für Informatik – Bibliothek – Postfach 6980 – 76128 Karlsruhe

Software-Industrialisierung

Herausgeber:
Franz Brosch, Thomas Goldschmidt, Henning Groenda,
Lucia Kapova, Klaus Krogmann, Michael Kuperberg,
Anne Martens, Christoph Rathfelder, Ralf Reussner,
Johannes Stammel

Autoren:
Jaouad Bouras, Benjamin Klatt, Maxim Kremer, Jörn
Kuhlenkamp, Igor Lankin, Jochen Maier, Yanjun Zhang

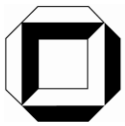
Interner Bericht 2008-8



Universität Karlsruhe

Fakultät für Informatik

ISSN 1432 – 7864



Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Software-Industrialisierung

Seminar im Sommersemester 2008

Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation
Lehrstuhl für Software-Entwurf und -Qualität
Prof. Dr. Reussner

http://sdq.ipd.uni-karlsruhe.de

Jaouad Bouras, Franz Brosch, Thomas Goldschmidt, Henning Groenda, Lucia Kapova, Benjamin Klatt, Maxim Kremer, Klaus Krogmann, Jörn Kuhlenkamp, Michael Kuperberg, Igor Lankin, Jochen Maier, Anne Martens, Christoph Rathfelder, Ralf Reussner, Johannes Stammel, Yanjun Zhang

Vorwort

Die Industrialisierung der Software-Entwicklung ist ein zurzeit sehr stark diskutiertes Thema. Es geht dabei vor allem um die Effizienzsteigerung durch die Steigerung des Standardisierungsgrades, des Automatisierungsgrades sowie eine Erhöhung der Arbeitsteilung. Dies wirkt sich einerseits auf die den Software-Systemen zu Grunde liegenden Architekturen aber auch auf die Entwicklungsprozesse aus. So sind service-orientierte Architekturen ein Beispiel für eine gesteigerte Standardisierung innerhalb von Software-Systemen. Es ist zu berücksichtigen, dass sich die Software-Branche von den klassischen produzierenden Industriezweigen dadurch unterscheidet, dass Software ein immaterielles Produkt ist und so ohne hohe Produktionskosten beliebig oft vervielfältigt werden kann. Trotzdem lassen sich viele Erkenntnisse aus den klassischen Industriezweigen auf die Software-Technik übertragen.

Die Inhalte dieses Berichts stammen hauptsächlich aus dem Seminar „Software-Industrialisierung“, welches sich mit der Professionalisierung der Software-Entwicklung und des Software-Entwurfs beschäftigte. Während die klassische Software-Entwicklung wenig strukturiert ist und weder im Bezug auf Reproduzierbarkeit oder Qualitätssicherung erhöhten Anforderungen genügt, befindet sich die Software-Entwicklung im Rahmen der Industrialisierung in einem Wandel. Dazu zählen arbeitsteiliges Arbeiten, die Einführung von Entwicklungsprozessen mit vorhersagbaren Eigenschaften (Kosten, Zeitbedarf, ...) und in der Folge die Erstellung von Produkten mit garantierbaren Eigenschaften. Das Themenspektrum des Seminars umfasste dabei unter anderem:

- Software-Architekturen
- Komponentenbasierte Software-Entwicklung
- Modellgetriebene Entwicklung
- Berücksichtigung von Qualitätseigenschaften in Entwicklungsprozessen

Das Seminar wurde wie eine wissenschaftliche Konferenz organisiert: Die Einreichungen wurden in einem zweistufigen Peer-Review-Verfahren begutachtet. In der ersten Stufe wurde eine Begutachtung der studentischen Arbeiten durch Kommilitonen durchgeführt, in der zweiten Stufe eine Begutachtung durch die Betreuer. In verschiedenen *Sessions* wurden die Artikel an zwei *Konferenztagen* präsentiert. Der beste Beitrag wurde durch einen *Best Paper Award* ausgezeichnet. Dieser ging an Benjamin Klatt für seine Arbeit *Software Extension Mechanisms*, dem hiermit noch einmal herzlich zu dieser herausragenden Leistung gratuliert wird. Ergänzend zu den Vorträgen der Seminarteilnehmer wurde ein eingeladener Vortrag gehalten. Herr Florian Kaltner und Herr Tobias Pohl vom IBM-Entwicklungslabor gaben dabei dankenswerterweise in ihrem Vortrag Einblicke in die Entwicklung von Plugins für Eclipse sowie in die Build-Umgebung der Firmware für die zSeries Mainframe-Server.

Gliederung

Die Themen dieses Seminars spiegeln verschiedenen Frage- und Problemstellungen wieder, die sich bei der Software-Industrialisierung ergeben. Dieser technische Bericht gliedert sich dabei wie folgt. Zunächst werden die Themen *Enterprise Application Integration* und *service-orientierte Architecture* vorgestellt und diskutiert. Im Folgenden wird auf die Spezifikation von Qualitätsanforderungen sowie die optimale Dimensionierung der verwendeten Hardware-Ressourcen eingegangen. Der nächste Beitrag behandelt das Thema *Architecture Centric Model-Driven Software Development*. Als nächstes folgt ein Überblick über verschiedene Erweiterungstechniken für Software. Der Beitrag des parallel zum Seminar laufenden Proseminars über das *Graphical Editing Framework* bildet den Abschluss dieses Berichts.

Dank

Wir möchten uns an dieser Stelle bei allen Teilnehmern des Seminars für ihre engagierte Mitarbeit sehr herzlich bedanken. Ein mehrstufiger Begutachtungsprozess bestehend aus Peer–Reviews sowie Gutachten durch die Betreuer ermöglichte die Auswahl qualitativ hochwertiger Artikel. Insgesamt wurden sieben Ausarbeitungen für diesen technischen Bericht angenommen. Auf der Homepage[1] zu diesem Seminar sind daneben auch die Vortragsfolien der Seminarteilnehmer zu finden, die auf den Sessions der Konferenz des Seminars vorgestellt wurden.

Ganz besonders möchten wir uns bei Herrn Florian Kaltner und Herrn Tobias Pohl aus dem IBM-Entwicklungslabor in Böblingen für ihren sehr interessanten Vortrag bedanken.

Karlsruhe, Juli 2008

Franz Brosch
Thomas Goldschmidt
Henning Groenda
Lucia Kapova
Klaus Krogmann
Michael Kuperberg
Anne Martens
Christoph Rathfelder
Ralf Reussner
Johannes Stammel

[1] http://sdqweb.ipd.uka.de/wiki/Seminar_Software-Industrialisierung_SS08

Inhaltsverzeichnis

Software-Industrialisierung

Enterprise Application Integration

Igor Lankin

Betreuer: Franz Brosch

Zusammenfassung Die vorliegende Arbeit gibt einen Überblick über das Thema der Enterprise Application Integration (EAI). Es werden die Probleme beleuchtet, die in IT-Infrastrukturen existieren und welche Lösungen EAI-Technologien dafür bereithalten. Konzepte der EAI werden, angefangen bei altbekannten Methoden wie verteilten Prozeduren und Objekten, über Middleware bis hin zu aktuellen dienstbasierten Ansätzen vorgestellt, wobei der Enterprise Service Bus genauer vorgestellt wird.

1 Einleitung

Obwohl Interoperabilität, Wiederverwendbarkeit und Erweiterbarkeit längst bekannte und geschätzte Eigenschaften von Software-Systemen sind, fanden sie in der Vergangenheit auf der Interapplikationsebene nicht immer die gewünschte Beachtung. Nichtsdestoweniger war man immer daran interessiert in Unternehmen eingesetzte Systeme zu verknüpfen und sowohl Daten als auch Funktionalität anwendungsübergreifend zu nutzen. Die geschaffenen Integrationslösungen erwiesen sich auf Grund von schnell gewachsenen IT-Infrastrukturen, fehlender Technologien oder schlichtweg unzureichender Planung nicht in vielen Fällen effektiv [Lint00, S. 6ff]. Die wachsenden Anforderungen an die Kommunikation und Kollaboration innerhalb und außerhalb von Unternehmen und somit an die dort eingesetzten Informationssysteme erfordern heutzutage zusätzliche, umfassende Maßnahmen und neue Lösungsstrategien auf diesem Gebiet. Die Disziplin, die sich mit dieser Problematik beschäftigt heißt “Enterprise Application Integration” (EAI). Seit mehreren Jahren werden unter diesem Begriff Konzepte und Technologien verstanden die als systematischer Ansatz zur Lösung von Integrationsszenarien in der Fachliteratur behandelt werden. [Lint00][Lint03][ACKM04]

Das Aufkommen der dienstorientierten Architekturen (engl.: service-oriented architecture, SOA) in den letzten Jahren und ihre derzeitige Popularität haben auch in der EAI neue Konzepte auf dieser Grundlage entstehen lassen. Lose Kopplung der Teilsysteme und die daraus resultierenden Vorteile, wie etwa Austauschbarkeit von Komponenten oder Applikationen, macht SOA für den Enterprise-Architekten interessant. Kommerzielle Hersteller und Open-Source-Organisationen bringen immer ausgereiftere Technologien auf den Markt, die die Idee der SOA in EAI unterstützen und Standards und Leitlinien entstehen lassen.

In dieser Arbeit wollen wir einen Überblick über bekannte “Enterprise Application Integration”-Konzepte und verfügbare Technologien bieten. Wir untersuchen welchen Problemen Unternehmen bei der Integration ihrer Informationssysteme gegenüberstehen und welche Lösungsansätze EAI für diese bietet. Vor

allem soll der neue Trend in Richtung dienstorientierter Architekturen und aktuell entstehende Technologien auf dieser Basis vorgestellt und untersucht werden.

Im Abschnitt 2 untersuchen wir, was genau unter dem Begriff der "Enterprise Application Integration" verstanden wird und stellen eine eigene Definition vor. Darauf folgen Ziele und Vorteile des Einsatzes von EAI in unterschiedlichen Integrationsszenarien sowie Probleme, die dabei auftreten können. Den Schluss dieses Abschnittes bildet dann ein Überblick über verschiedene Klassifikationsansätze von Integrationsebenen.

Abschnitt 3 bietet einen Einblick in bekannte und seit langem eingesetzte EAI-Konzepte und Technologien. Es wird beschrieben inwiefern diese Technologien die Integration auf unterschiedlichen Ebenen ermöglichen und welche Rolle sie mit oder neben dienstorientierten Ansätzen spielen. Weiter folgt die Vorstellung der Ideen der dienstorientierten Architekturen und einiger Technologien auf dieser Basis. Der "Enterprise Service Bus" findet dabei im Abschnitt 3.4 besondere Beachtung.

2 Grundlagen

Versuche festzulegen was unter EAI zu verstehen ist, fallen in der Literatur oft eher deskriptiv aus. Kaum findet sich ein Versuch den Begriff in einer Definition zusammenzufassen. (Vergleiche [ACKM04], [Chap04], [CHKT06], [Lint00], [Kell02]). Das liegt nicht nur daran, dass das Thema sehr vielschichtig ist und die Fokussierung auf bestimmte Herangehensweisen zu verschiedenen Auffassungen führt. Es variiert auch die Art und Weise wie eingesetzte Technologien, Standards, Soft- und Hardwarelösungen, Architekturen, Geschäftsprozessmodelle und "Best Practices" unter dem Begriff "EAI" zusammengefasst werden. So wird in [ACKM04, Kap. 3.1.] etwa zwischen *Middleware* (siehe Abschnitt 3.1) und EAI differenziert. Andere sehen in der Integration mittels Middleware bereits einen Ansatz der zur Enterprise Application Integration gezählt werden kann [Lint00], [CHKT06].

Im Vorwort von [Lint00] beschreibt David Linthicum EAI als "the nexus of technology, method, philosophy, and desire to finally address years of architectural neglect". Obwohl diese Zusammenfassung recht einschwörend klingt, und angesichts des damaligen Stellenwerts von EAI als aufkommende Disziplin vielleicht angebracht war, möchten wir unsere Definition in Anlehnung daran formulieren.

Definition 1 (Enterprise Application Integration). *Unter* Enterprise Application Integration (EAI) *wird der* fortwährende *Vorgang der Integration von Geschäftsprozessen in und zwischen Unternehmen, sowie die Gesamtheit der dabei eingesetzten Methoden, Technologien und Werkzeuge verstanden.*

2.1 Motivation von EAI

Die Informations- und Telekommunikationstechnologie entwickelt sich so rasant wie kaum ein anderer Industriezweig. Softwarelebenszyklen werden immer kürzer

und die Vielzahl an verfügbaren Informationssystemen macht es Unternehmen schwierig zu entscheiden welche Lösungen sich für den Einsatz eignen. Nicht selten fällt die Entscheidung zu Gunsten der neuesten und populärsten Produkte und Technologien aus, ohne genau zu untersuchen wie sie sich in die vorhandene IT-Infrastruktur integrieren lassen. Die Entscheidungen werden häufig nach der "Best of Breed"-Strategie, auf Abteilungsebene, unabhängig von den Integrationsmöglichkeiten getroffen [Lint00]. Kombiniert mit schwacher architektonischer Voraussicht, oder gar gänzlich unkontrolliertem Wachstum der Unternehmens-IT, sind weitgehend heterogene Teilsysteme und Einzellösungen ("accidential architecture" [Chap04], siehe Abbildung 1) entstanden, die einen enormen manuellen Aufwand und redundante Vorgänge benötigen um den globalen Geschäftsprozessen gerecht zu werden.

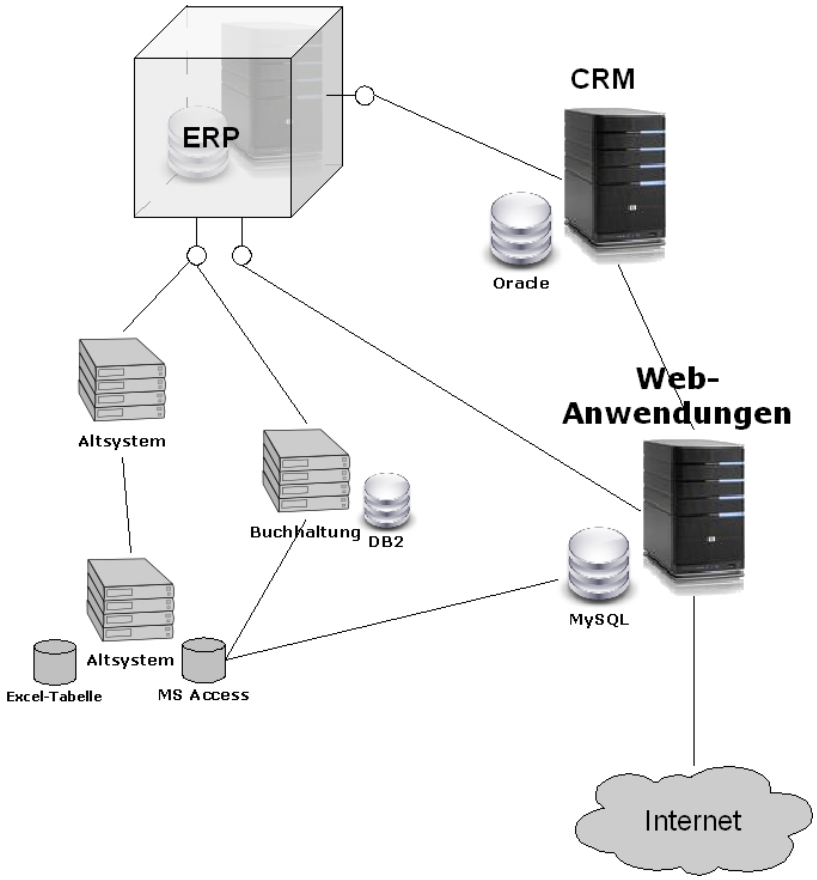


Abbildung 1. Gewachsene Architektur mit Punkt-zu-Punkt-Integration (vgl. [Lint00, S. 9])

Die Idee von “Straight-Through Processing” (STP), ist es genau solche ineffizienten Abläufe und die damit verbundenen Kosten zu eliminieren. Jedes Datum soll innerhalb eines Geschäftsprozesses genau ein Mal vorliegen und möglichst automatisch oder mit wenig Aufwand verarbeitet werden können. Die Dauer von einzelnen Vorgängen wird auf ein Minimum reduziert. Der “Hype” um E-Commerce zwang selbst traditionelle Unternehmen am Internetboom teilzunehmen und sich Gedanken um STP, und damit auch um EAI, zu machen. Das Aufkommen von Online-Marktplätzen und verschiedenen Business-to-Business (B2B)-Modellen verlangte von den Unternehmen ihre Geschäftsprozesse grundlegend zu überdenken um weiterhin wettbewerbsfähig bleiben zu können. Die Integration der benutzen Applikationen ist für STP eine Voraussetzung, was STP zu einem der stärksten Treiber von EAI macht. [Chap04]

Die Ausgaben für IT sind jedoch seit der Jahrtausendwende stetig gesunken. Finanzielle Mittel stehen nur für kritische Systeme zur Verfügung. Wo früher ältere Informationssysteme durch neue mächtige Komplettlösungen ersetzt werden konnten, haben IT-Verantwortliche heutzutage häufiger die Aufgabe weiterhin mit den Altsystemen zu arbeiten und diese zu integrieren.

Als Altsysteme (legacy system), werden diejenigen Informationssysteme bezeichnet, die seit längerer Zeit im Produktivsystem des Unternehmens eingesetzt werden, jedoch vom Stand der Technik als veraltet gelten, weil etwa eine neue Version des Systems oder sogar eine andere Technologie existiert. Der Ersatz eines Altsystems ist in vielen Fällen ein kritischer Faktor für den Produktivbetrieb eines Unternehmens. Eine Umstellung des Systems, die in der Regel mit einer Unterbrechung des laufenden Betriebs einhergeht, ist daher risikoreich und teuer. Zum einen entstehen Kosten durch den Ausfall selbst und zum anderen muss das Entwickeln und/oder Aufsetzen der neuen Systeme von strengen Qualitässicherungsmaßnahmen begleitet werden. Dazu kommen dann Folgekosten der Umstellung, die etwa durch die Schulung der Mitarbeiter und die herabgesetzte Produktivität der Benutzer in der Anlaufzeit entstehen. Nach der Abwägung aller Risiken und Umstände, stellt sich in vielen Fällen heraus, dass eine Umstellung nicht zu rechtfertigen ist. [Lint00]. Somit müssen diese Systeme weiterhin verwendet werden und Techniken der EAI genutzt werden um sie zu integrieren.

Problematisch wird die Integration von Altsystemen, wenn die Weiterentwicklung der Software nicht oder nur zum Teil möglich ist. Eine Erweiterung ist oftmals nötig um Schnittstellenanpassungen durchzuführen. Liegt aber zum Beispiel der Quellcode nicht vor oder wäre die Einarbeitung und Weiterentwicklung des Codes zu aufwendig, so müssen alternative Lösungen geschaffen werden.

Ähnliche Probleme gibt es aber auch bei fertigen Softwarepaketen wie “Enterprise Resource Planing”-Systemen (ERP-Systeme). Während diese Produkte einen vollen Umfang von Lösungen für eine Reihe von Geschäftsprozessen liefern, sind die internen Daten und Vorgänge nicht oder nur sehr begrenzt für eine externe Weiterverarbeitung verfügbar. Und obwohl neuere Fertigsysteme Ansätze zur erhöhten Interoperabilität aufweisen, bleiben zugekaufte Komplettlösungen eine der am schwierigsten zu integrierenden Anwendungen und damit eine treibende Kraft für Enterprise Application Integration.

Das Ziel von EAI ist es diese bestehenden Informationssysteme zu integrieren, sodass Informationsflüsse rationalisiert und beschleunigt werden. Statt unabhängiger Punkt-zu-Punkt-Integration wird eine Gesamtarchitektur verwendet, die mit Hilfe von unterschiedlichen Integrationstechnologien, den Datenfluss in den Geschäftsprozessen entsprechend koordiniert (siehe Abbildung 2). Es wird eine Infrastruktur geschaffen, in der Daten und Funktionalität der unterschiedlichen Teilsysteme einheitlich verwendet werden können. Applikationen, die neu hinzukommen können so einfach in die Umgebung eingebunden werden. Desweiteren können Daten und Dienste des vorhandenen Systems über Unternehmensgrenzen hinweg zur Verfügung gestellt werden. Auf diese Weise kann eine Automatisierung von Abläufen zwischen Unternehmen, wie zum Beispiel in Wertschöpfungsketten, ermöglicht werden.

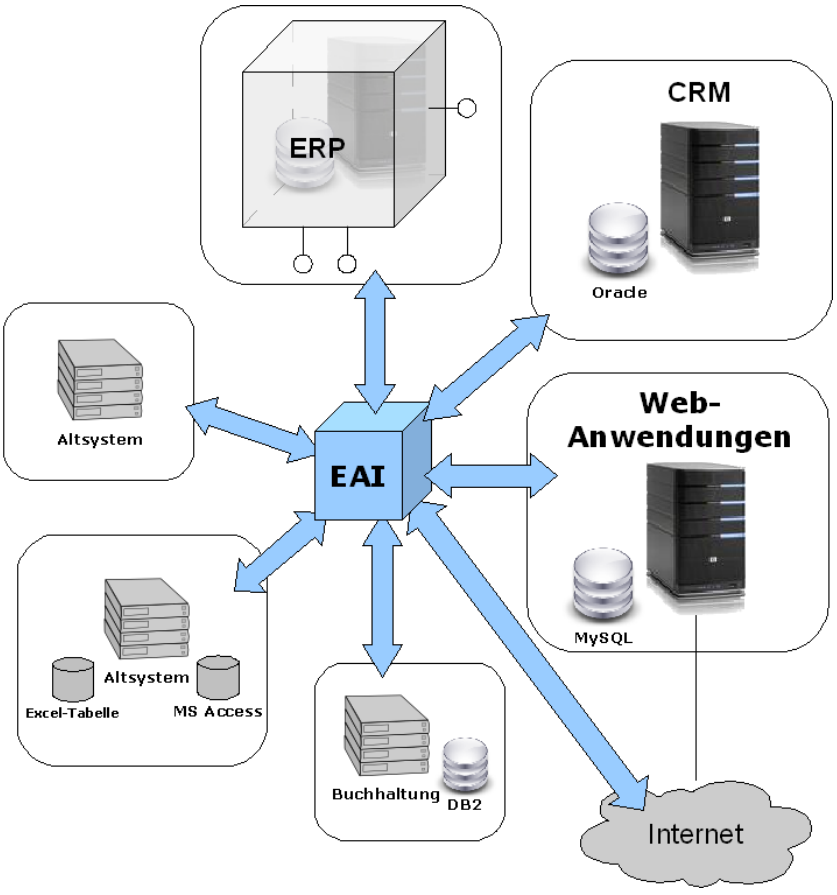


Abbildung 2. EAI-Lösung (vgl. [Lint00, S. 10])

2.2 Arten der Integration

Die Enterprise Application Integration kann auf verschiedenen Ebenen stattfinden. Abhängig vom Anwendungsfall findet die Integration auf

- Datenebene,
- Prozessebene oder
- Präsentationsebene

statt.

Die Integration auf der Datenebene ist eine der einfachsten und gleichsam effizientesten Methoden der EAI. In vielen Fällen müssen Datenbestände abgeglichen werden oder von verschiedenen Stellen abgefragt werden. Dabei muss berücksichtigt werden, dass die Schemata, Formate und Semantik der Daten in unterschiedlichen Anwendungen und Datenbanken verschieden sein können. So können zum Beispiel Unterschiede zwischen den Datentypen auftreten, Datenskalierungen können variieren (z.B. unterschiedliche Währungen und Einheiten) und es können Felder einer Datenquelle in einer anderen über mehrere Felder und Tabellen verteilt sein. Die Vorteile von einer Integration auf dieser Ebene ist zum einen, dass sehr viele gut verstandene EAI-Technologien und ausgereifte Werkzeuge existieren um Daten zu konvertieren, aggregieren, synchronisieren und einheitlich zur Verfügung zu stellen. Zum anderen brauchen bestehende Anwendungen nicht angepasst werden, was vor allem in kritischen Anwendungen mit hohen Kosten und Risiken verbunden wäre. Jedoch können Datenbankinhalte durch Normalisierung und technische Modifikation der Daten enorm kompliziert sein (z.B. Verteilung auf mehrere Tabellen, Internationalisierung, etc). Auch die Aggregation der Daten, der eine bestimmte Geschäftslogik unterliegt, und in bestehenden Applikation implementiert ist, wird bei der Integration auf der Datenebene nicht genutzt. [CHKT06][Lint00]

An dieser Stelle setzt die Integration auf Prozessebene an. Programmierschnittstellen (Application Programming Interface, API) und Dienste bestehender Applikationen werden verfügbar gemacht um sowohl Daten als auch Funktionalität wieder zu verwenden. Heterogene Technologien und unterschiedliche Qualitäten und Umfänge von APIs führen jedoch zu Inkompatibilitäten und Problemen bei der Integration. Schnittstellen müssen in bestehenden Altsystemen nachgerüstet oder modifiziert werden. Der Aufwand für Programmierung und Qualitätskontrolle, der hier entsteht, kann enorm sein. Und während einige Fertigsysteme offene Schnittstellen unter Ausnutzung von standardisierten Technologien zur Verfügung stellen, bieten viele andere proprietäre APIs an, die nur begrenzt (zum Beispiel nur in bestimmten Programmiersprachen) verwendet werden können.[Lint03][CHKT06]

EAI auf der Präsentationsebene ist die primitivste Form der Integration. Benutzerschnittstellen werden dafür verwendet Daten und Funktionalität aus der Anwendung zu extrahieren (“Screen Scraping”). Dafür wird die Interaktion eines Benutzers durch Programmcode simuliert. In der Ausgabe auf der Benutzerschnittstelle werden die benötigten Informationen lokalisiert und extrahiert. Diese Art von Integration geschieht meist in den Fällen, in denen ein Zugriff

auf Daten- oder Prozessebene nicht möglich oder unverhältnismäßig schwieriger ist. Der Prozess der Informationsextraktion aus Web-Seiten oder textbasierten Schnittstellen gestaltet sich durch die Einfachheit des Formates einfacher als aus graphische Benutzerschnittstellen, z.B. Dialogfenstern einer Windows-Anwendung. Auch existieren mehrere Werkzeuge die "Screen Scraping" vor allem für Webseiten unterstützen. Jedoch skaliert dieser Ansatz generell schlecht und kann allgemein als instabil bezeichnet werden. [Lint00][Kell02]

In unserer Arbeit wollen wir uns ausschließlich mit der Integration auf Prozessebene beschäftigen, und während die Datenintegration dabei oft auch eine wichtige Rolle spielt, werden wir die Integration über Benutzerschnittstellen nicht weiter untersuchen.

3 Konzepte und Technologien

Eine Vielzahl von Konzepten der Enterprise Application Integration ist über die Zeit entstanden. Und obwohl ununterbrochen Wandlungen und Innovationen in den Technologien zu verzeichnen sind, bestehen die Grundideen oft seit mehreren Jahrzehnten. In diesem Abschnitt stellen wir einige dieser Technologien und Konzepte vor. Nach einem Überblick über die Basistechnologie der EAI, der Middleware, untersuchen wir Methoden der Enterprise Application Integration mit SOA.

3.1 Middleware als Basis der EAI

Jede Art von System, die eine Kommunikation zwischen unterschiedlichen Applikationen ermöglicht kann grundsätzlich als Middleware bezeichnet werden. Es gibt die unterschiedlichsten Techniken für die Verteilung von Enterprise-Softwarekomponenten. Wie wir bereits beschrieben haben besteht eine starke Heterogenität zwischen den Informationssystemen eines Unternehmens und somit der verfügbaren Schnittstellen und Kommunikationsformen.

In den Anfängen der verteilten Programmierung wurde die Kommunikation zwischen zwei verteilten Anwendungen direkt implementiert, basierend auf dem reinen physischen Netzwerkprotokoll. Die Entwickler mussten sich um die exakten Details des Datenaustausches über das Netzwerk kümmern. Netzwerkpakete mussten erstellt, gesendet und empfangen werden. Sehr viel Aufwand fiel daher auf die technischen Aspekte in solchen Softwaresystemen. Die Kommunikation war an die entsprechenden Protokolle angepasst und machte eine Erweiterung auf andere Netzwerktypen inpraktikabel.

Remote Procedure Calls (RPC) wendeten das Konzept des lokalen Prozeduraufrufs auf verteilte Anwendungen an. Eine lokale Funktion oder Prozedur kapselt einen Programmcodeabschnitt und macht ihn auf der entfernten Seite wiederverwendbar. Der Programmcode kann dann auf der entfernten Seite wie eine normale Prozedur aufgerufen werden, in dem der Aufruf über das Netzwerk an die Quellanwendung weitergeleitet wird, die die Berechnung durchführt und

das Ergebnis wiederum über das Netzwerk zurückliefert. Dabei passiert der Aufruf in den meisten RPC-Implementierungen synchron, was bedeutet, dass der Prozess auf dem Client so lange blockiert wird, bis der Server die Anfrage beantwortet hat, weshalb RPC auch als "blockierende Middleware" bezeichnet wurde. Die Begriffe Server und Client werden hier lediglich zur Identifizierung der Seiten verwendet, die respektive eine entfernte Prozedur zur Verfügung stellen oder diese aufrufen. Diese gelten nur für einen einzelnen Prozeduraufruf, da die Seiten auch gegenseitig Programmcode abrufen können. Entfernte Prozeduren können über RPC also wie lokale Prozeduren aufgerufen werden. Vertreterprozeduren (Proxys oder Stubs) bilden die entfernte Schnittstelle lokal ab und verbergen die Netzwerkommunikation. [Lint00]

Die Entwicklung der objektorientierten Programmierung am Anfang der 90er Jahre erweiterte die Idee des entfernten Prozeduraufrufs auf Objekte entsprechend dem OO-Paradigma. Unter Einsatz von sogenannten Object Request Broker (ORB) können Objekte über plattformunabhängige Protokolle (heutzutage meist SOAP) über das Netzwerk kommunizieren. Die bekanntesten ORB-Implementierungen sind CORBA (Common Object Request Broker Architecture), COM/DCOM und RMI. Während RMI auf Java und COM/DCOM auf Microsoft-Plattformen beschränkt sind, deckt CORBA zahlreiche Plattformen und Programmiersprachen ab. [Kell02][Lint00][Chap04]

Die Integration von mehreren Anwendungen mittels verteilten Prozeduraufrufen oder verteilten Objekten ermöglicht die Bereitstellung von Daten und Funktionalität zwischen verteilten Applikationen. Aufrufe können aus dem Programmcode direkt an die Quelle abgesetzt werden und erhalten dank der Synchronität direkt ein Ergebnis. Jedoch hat der direkte Zugriff auf entfernten Programmcode eine starke Kopplung zwischen den Anwendungen zur Folge. Wird ein Aufruf über eine Reihe von verteilten Prozessen durchgeführt, so hängt dessen Erfolg von der Erreichbarkeit aller in der Aufrufabfolge beteiligten Prozeduren ab. Ist die Kommunikation zu einem beteiligten Prozess unterbrochen, so wird der gesamte Aufruf und alle Aufrufe, die von diesem abhängen, fehlschlagen (siehe Abbildung 3).

Nachrichtenorientierte Middleware (Message Oriented Middleware, MOM) ist eine Technologie, die die starke Kopplung von RCP-basierten Architekturen vermeidet. Daten werden in Kommunikationskanälen in Form von abgeschlossenen Informationseinheiten (Nachrichten) übermittelt. Üblicherweise passiert die Übertragung asynchron. Nach dem Absetzen einer Nachricht kann ein Verarbeitungsprozess fortgesetzt werden und falls eine Antwort erwartet wird, kann diese zu einer gegebenenfalls günstigeren Zeit abgefragt werden. Durch den Einsatz von MOM-basierter Integration werden die Anwendungen von einander entkoppelt. Sender und Empfänger müssen sich nicht direkt kennen, um Daten auszutauschen. Stattdessen senden und empfangen sie Nachrichten über ein Vermittlungssystem (Kommunikationsserver) wie in Abbildung 4 dargestellt. Das Vermittlungssystem stellt sicher dass die Informationen korrekt zugestellt werden. [KrBS04, 3.2][Chap04, Kap. 5][Lint00, Kap. 7]

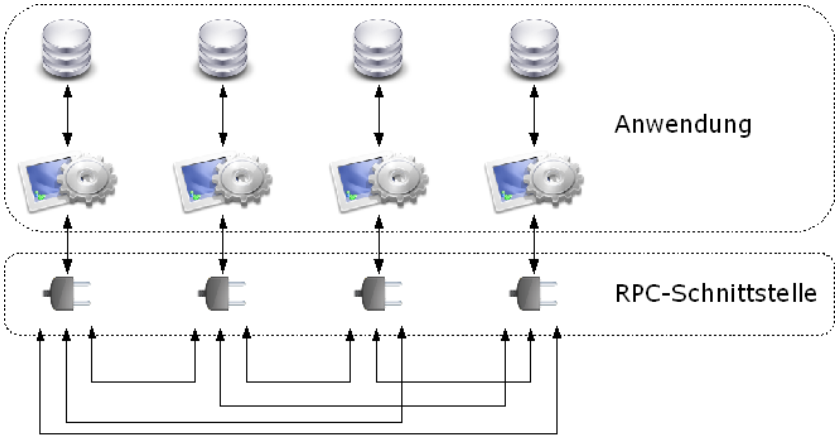


Abbildung 3. Synchrone, verteilte Kommunikation besteht aus einer Reihe von Punkt-zu-Punkt Verbindungen, die von einander abhängen können, vgl. [Chap04, Kap. 5]

Auch wenn die Komplexität MOM-basierter Lösungen wegen der Asynchronität im Vergleich von synchroner Integration mit verteilten Objekten und Prozeduren in der Regel höher ist, überwiegen in den meisten Fällen die Vorteile der losen Kopplung und zentralen Verarbeitungsmöglichkeiten. Die unterschiedlichen Datenformate, die jede einzelne Applikation bietet können durch Transformation im Kommunikationsserver in wiederverwendbare, auf Standard gestützte Formate umgewandelt werden. Man ist auf diese Weise nicht gezwungen jedem Nachrichtenempfänger die Verständigung mit jedem potenziellen Sender "beizubringen".

Die Vorteile der Integration mit Hilfe von MOM haben jedoch auch ihren Preis. Bestehende Anwendungen müssen an das Kommunikationssystem angepasst werden. Weite Teile von Altsystemen müssen gegebenenfalls neu entwickelt oder stark verändert werden um die asynchrone Nachrichtenbehandlung zu unterstützen. Anbieter von Fertigsystemen wie ERP und Middlewaresysteme bieten zwar oft Adapter-Lösungen für MOM, es bleibt aber sehr viel Konfigurations-, Entwicklungs- und Testaufwand um die vollständige Integration zu bewerkstelligen.

Nichtsdestoweniger ist die nachrichtenorientierte Middleware ein sehr wichtiges Konzept für die EAI. Der sogenannte Enterprise Service Bus (ESB), den wir im Abschnitt 3.4 näher beschreiben, basiert zum Beispiel auf genau diesem Konzept. [Lint00][Chap04][Kell02]

Transaktionsmonitore (auch Transaction Processing (TP)-Monitore) sind eine der ältesten Formen von Middleware. Bereits Ende der 60er Jahren wurde IBMs Customer Information Control System (CICS) entwickelt [Wiki08b].

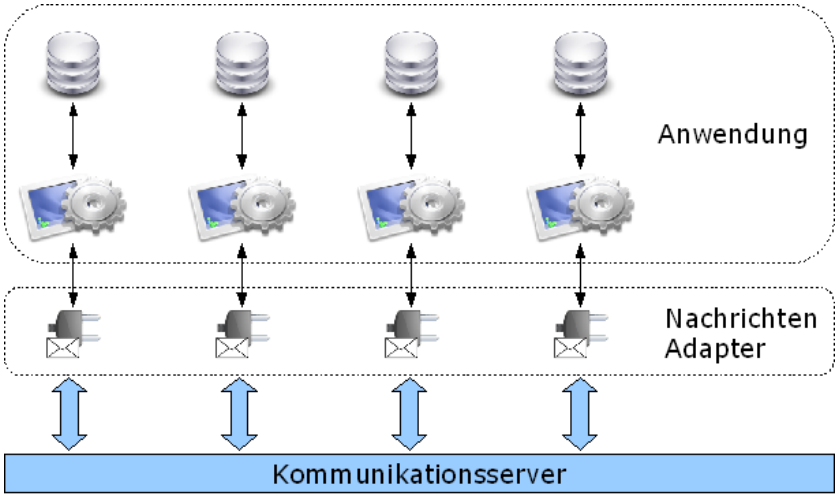


Abbildung 4. Einsatz eines Kommunikationsservers vgl. [Chap04, Kap. 5]

Transaktionsmonitore bieten die Möglichkeit Anwendungen auszuführen, die sehr viele Benutzer gleichzeitig bedienen. Sie garantieren, dass Anwendungen auch bei Höchstlast transaktional und möglichst performant abgewickelt werden können. Durch Herunterbrechen von einzelnen Vorgängen in kleinere Aufgaben, werden diese als Transaktionen durchgeführt. Das stellt zum einen sicher, dass bei hoher Last die Datenintegrität in den abgefragten Ressourcen gewährleistet ist, und erlaubt zum anderen die Anwendung von Mechanismen zur Lastverteilung. [Lint00]

3.2 Dienstorientierte EAI

Dienstorientierte Architekturen (Service-Oriented Architectures, SOA) haben sich zu einer mächtigen Strategie entwickelt, verteilte Anwendungen herzustellen. Die Idee ist es eine Architektur zu erstellen, die es ermöglicht lose gekoppelte, transparent lokalisierbare, standardbasierte, und plattformunabhängige "Dienste" zur Verfügung zu stellen. Die Idee eines Dienstes existiert in der Softwareentwicklung schon sehr lange, jedoch werden darunter unterschiedliche Konzepte verstanden. Entfernt zur Verfügung stehende Prozeduren, wie wir sie im Kapitel 3.1 beschrieben haben, werden manchmal als Dienste bezeichnet. Auch im SOA-Umfeld gibt es teilweise unterschiedliche Auslegungen des Dienstbegriffes genauso wie des SOA-Begriffes selbst. In einer SOA verstehen wir unter einem *Dienst* oder Service ein in sich abgeschlossenes Stück Software, dass über ein Netzwerk abgerufen werden kann und eine genau definierte Geschäftslogik implementiert. Dafür muss der Dienst eine Dienstbeschreibung inklusive einer wohldefinierten Schnittstelle festlegen und vom Kontext und Zustand anderer Dienste unabhängig sein. Die Dienstbeschreibung kann neben der Beschreibung

der Schnittstelle, über die der Dienst anzusprechen ist, weitere Informationen enthalten. So werden etwa auch notwendige Verbindungsdaten beschrieben und definiert welche Protokolle für die Kommunikation verwendet werden. Darüber hinaus können Informationen zur im Dienst verwendeten Ontologie und Taxonomie oder zum Unternehmen, welches den Dienst bereitstellt, veröffentlicht werden.[PaHe07][KrBS04][ACKM04]

Ein Service Bus ist die Integrationsplattform für Services. Er wird eingesetzt um die Kommunikation zwischen Diensten und Dienstkonsumenten zu bewerkstelligen und eine Reihe von Aufgaben wie zum Beispiel Qualitätssicherung, Protokollierung oder Sicherheitsmechanismen zu übernehmen. Im Unterschied zur Middleware ist der Service Bus keine einzelne zentrale Komponente, sondern besteht aus einer Vielzahl von verteilten Komponenten und wird durch den Einsatz verschiedener Technologien realisiert. In Abschnitt 3.4 stellen wir den Enterprise Service Bus (ESB) als ein, neues Konzept des Services Bus' vor, der sich vor allem für große heterogene Infrastrukturen und somit für den Einsatz in der EAI eignet.

Darüber hinaus erlaubt das Konzept eines Verzeichnisdienstes (Service Repository) eine dynamische Veröffentlichung und Nutzung von Diensten. Einem Dienstbenutzer muss somit nicht vorab mitgeteilt werden welche Dienste existieren und wie sie erreichbar sind. Stattdessen wird eine Sammlung verfügbarer Dienste verwaltet. Veröffentlicht ein Dienstanbieter einen Dienst, so haben (potenzielle) Dienstkonsumenten die Möglichkeit über das Service Repository seine Beschreibung abzurufen und den Dienst zu nutzen.

Die Dienstinfrastruktur in SOA versucht so viele technische Details wie möglich von den Dienstkonsumenten zu verbergen. Ob ein Dienst in J2EE oder .NET umgesetzt ist, hat keinerlei Auswirkungen auf den Benutzer. Auch erhält der Benutzer die Möglichkeit ohne nennenswerten Aufwand auf eine andere Dienstimplementierung zu migrieren, die etwa eine höhere "Dienstgüte" aufweist. [BrJK03][CHKT06]

Für die Enterprise Application Integration stellt SOA eine wichtiges architektonisches Konzept bereit um eine geschäftsprozessorientierte Gesamtlösung zu schaffen. Der Hauptfokus liegt darauf Dienste für alle Geschäftsvorgänge unternehmensweit und über Unternehmensgrenzen hinaus zur Verfügung zu stellen statt technische Aspekte, wie etwa einen Abruf von Daten aus einer Datenbank, zu kapseln. Die komplexe Anwendungslandschaft wird so in klar gegliederte Einheiten zerlegt und bietet allen Beteiligten einen eindeutig definierten Kontext für die Zusammenarbeit.

SOA ermöglicht es vorhandene Technologien weiterhin zu nutzen in dem, diese als Dienste gekapselt werden. Teure Neuentwicklungen können auf diese Weise vermieden werden. Der nachträgliche Austausch einer als Dienst gekapselten Anwendung kann wesentlich einfacher von statten gehen, da der Dienst auf der Ebene der Geschäftslogik benutzt wird und Implementierungsdetails verborgen sind.

Diensttypen Entsprechend der Klassifikation der Integrationsebenen können unterschiedliche Typen von Diensten identifiziert werden (hier nach [KrBS04, Kap. 5]), die in der EAI mit SOA vertreten sein können.

Basisdienste sind solche, die grundlegende Funktionen kapseln. Sie werden vor allem für die Bereitstellung von (aggregierten) Daten eingesetzt, die zur Weiterverarbeitung durch Anwendungen oder anderen Diensten genutzt werden können. Bei datenzentrierten Diensten werden dabei einzelne Geschäftseinheiten gekapselt statt einer Datenzugriffsschicht für die gesamte Anwendung zu abstrahieren. Auch komplexe Berechnungsroutinen eines Geschäftsfalls können in Basisdiensten umgesetzt werden.

Zwischendienste können für verschiedene Zwecke eingesetzt werden. Sie dienen als "Brücken" zur Überwindung von Technologielücken und Formatunterschieden, in dem sie als eine Art Übersetzer zwischen Systemen fungieren, auf denen inkompatible Technologien eingesetzt werden ("Technologiegateway"), oder die zur Kommunikation verschiedene Formate und Datenschemata benutzen ("Adapter"). Sogenannte "Fassaden" vereinfachen den Zugriff auf komplexe Anwendungen, in dem sie eine vereinfachte Schnittstelle anbieten, die auf den entsprechenden Geschäftsfall zugeschnitten ist. Und schließlich erweitern "Funktionalitätsergänzende Dienste" (analog dem Entwurfsmuster "Dekorator" der "Gang of Four") bestehende Dienste um zusätzliche Funktionalitäten.

Prozesszentrierte Dienste kontrollieren und verwalten den Status von Vorgängen auf höheren Ebenen der Geschäftsabläufe. Im Unterschied zu Zwischendiensten sind diese Dienste statusbehaftet, da ihre Funktionsweise vom Status des verwalteten Prozesses abhängt. Für die Entwicklung weiterer Komponenten erleichtern sie den Umgang mit der Geschäftslogik, in dem sie die Komplexität eines Vorgangs kapseln. Sie bieten Möglichkeiten zur Implementierung von Lastverteilungsmechanismen und weiterer Anforderungen an die Prozesssteuerung.

Öffentliche Unternehmensdienste werden im Gegensatz zu den zuvor genannten Diensttypen über die Unternehmensgrenzen hinaus angeboten. Kunden und Partner können somit in die übergeordneten Geschäftsprozesse einbezogen werden. Zum Beispiel können Geschäftsprozesse einer Wertschöpfungskette durch die Nutzung öffentlicher Unternehmensdienste enorm beschleunigt werden. Jedoch werden spezielle Anforderungen an solche Dienste gestellt. Zum einen müssen Schnittstellen eine Granularität auf der Ebene von Geschäftsdokumenten haben, was bedeutet, dass sie den vollständigen Kontext der gelieferten Daten beinhalten müssen um genutzt werden zu können, da zusätzliche Informationen (etwa aus anderen Diensten) möglicherweise nicht zur Verfügung stehen. Zum anderen müssen Sicherheitsmaßnahmen getroffen, Zugriffskontrollen erstellt und Abrechnungssysteme für die Dienstnutzung geschaffen werden.

Dienstorientierte Architekturen stellen somit ein mächtiges Konzept für die EAI dar. Sie ermöglichen eine lose Kopplung von Anwendungen, was eine verbesserte Erweiterbarkeit und Wiederverwendbarkeit zur Folge hat. Die Umsetzung von Diensten im Hinblick auf bestimmte Geschäftsabläufe erlaubt die Konzentration der Integration auf Ebene von Geschäftslogik und der organisatorischen Strukturen, anstelle von technischen Verknüpfungen von Softwaresystemen. Je-

doch ist der Einsatz von SOA insbesondere als EAI-Lösung zur Zeit noch mit hohen Risiken verbunden. Da das Konzept noch sehr neu ist, fehlt es zum einen an Erfahrung besonders in sehr großen Projekten wie etwa EAI. Zum anderen haben die Standards und unterstützenden Technologien noch keinen verlässlichen Reifegrad erreicht. [Gras05]

3.3 Web Services

Oft wird der Begriff "Web Service" mit Service gleichgesetzt. Dabei besteht die Idee der SOA zunächst unabhängig von der eingesetzten Technologie. Die meisten Dienste werden jedoch als Web Services implementiert. Zum einen trägt sicherlich die Verwendung der Web-Standards und deren Verbreitung zum Erfolgt von Web Services bei. Zum anderen wird die Entwicklung von Web Services durch Technologieplattformen wie .NET und J2EE stark unterstützt. Vor allem für einfachere Dienste eignen sich Web Services sehr gut. Wir wollen in diesem Kapitel einige Aspekte von Web Services im Hinblick auf ihren Einsatz in der EAI, insbesondere über Unternehmensgrenzen hinweg (Business-to-Business- oder B2B-Integration), untersuchen.

Die Mittel herkömmlicher Middleware-Ansätze reichen in der Regel nicht aus um die zusätzlichen Anforderungen der B2B-Integration zu erfüllen. Zwar ist von der technischen Seite eine Implementierung des Austausches von Funktionalität und Daten grundsätzlich möglich, jedoch gibt es zwischen verschiedenen Unternehmen organisatorische Aspekte die Probleme aufwerfen. Der Einsatz zentralisierter Lösungen durch Middleware für solche Szenarien wirft die Frage auf, auf welcher der beteiligten Seiten diese umgesetzt werden sollen. In der Regel sind die Beteiligten nicht bereit die Verarbeitung der Gegenseite oder einer dritten Partei zu überlassen. Zum einen wollen sie ihre Unabhängigkeit beibehalten, zum anderen kann Mangel an Vertrauen und die Vertraulichkeit von Daten und Geschäftsprozessen eine Rolle spielen.

Wie wir in dem vorhergehenden Abschnitt festgestellt haben, eignen sich öffentliche Unternehmensdienste für die Lösung verschiedener Szenarien der B2B-Integration wie etwa in Abbildung 5 dargestellt. Bestehende Applikationen oder Teile davon werden als Dienst gekapselt und über das Internet nach außen angeboten. Durch die Verwendung von standardisierten Web-Technologien, nämlich HTTP, XML und SOAP, sind Web Services als Implementierung solcher Dienste prädestiniert. Für die Dienstbeschreibung wird die sogenannte *Web Service Definition Language* (WSDL) verwendet. Im XML-Format werden in einer WSDL-Beschreibung die Schnittstelle, die Verbindungsoptionen und Protokolle des Dienstes definiert. In den meisten Fällen werden HTTP als Transportprotokoll und SOAP für die entfernten Aufrufe verwendet. Anders als oft angenommen, können auch andere Protokolle benutzt werden. Die Standardisierungsgrade hinter HTTP und SOAP, deren Verbreitung und die Unterstützung durch Applikationsplattformen und Entwicklungsumgebungen machen diese jedoch zur bevorzugten Wahl. Darüber hinaus erspart der Transport per HTTP Probleme, die durch Firewall-Konfigurationen entstehen. In den meisten Netzwerken sind Verbindungen am Standard-Port für HTTP (80) nicht beschränkt.

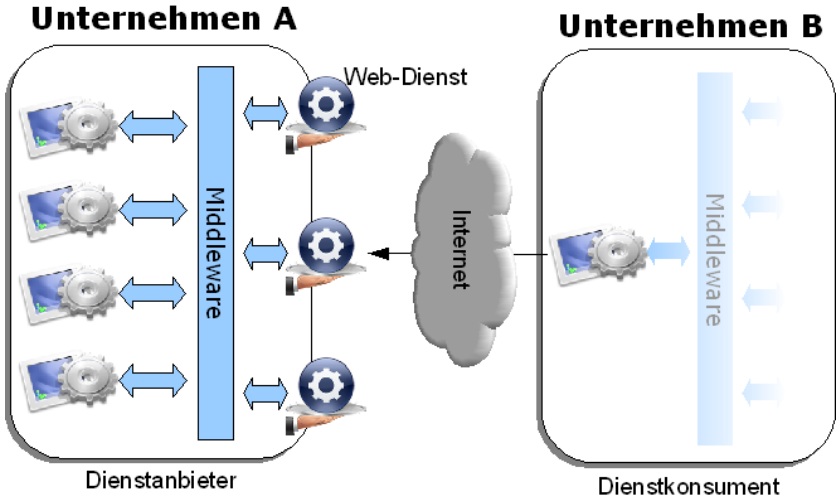


Abbildung 5. B2B-Integration über Webservices

Zusätzlich wurde ein Standard für XML-basierte Service Repositories eingeführt, *Universal Description, Discovery and Integration* (UDDI). UDDI-Server erlauben es Dienstanbietern ihre Dienste samt Dienstbeschreibungen sowie Informationen zum Unternehmen zu publizieren. Dienstkonsumenten können Suchanfragen an den Server richten und erhalten ggf. Informationen zu den benötigten Diensten. [CHKT06, 6.5]

Über diese grundlegenden Konzepte hinaus müssen besonders in dienstorientierten Architekturen, die mehrere Unternehmen umspannen, weitere Maßnahmen getroffen werden um den Anforderungen der öffentlichen Unternehmensdienste gerecht zu werden. Neben notwendigen Mechanismen zur Gewährleisung von Daten- und Prozessintegrität (Transaktionen) sowie zur Authentifizierung von Dienstbenutzer müssen auch unterschiedliche Dienste koordiniert und kombiniert werden können. Seit einiger Zeit existieren mehrere Vorschläge zur Lösung dieser Probleme. (Siehe auch Abbildung 6)

Die *Business Process Execution Language for Web Services* (BPELWS oder WS-BPEL), das durch die *Organization for the Advancement of Structured Information Standards* (OASIS) spezifiziert wurde, ist eine Sprache mit der das Zusammenspiel von Diensten auf der Ebene von Geschäftsprozessen formal beschrieben werden kann. Eine Geschäftsprozessdefinition umfasst die Beschreibung der Kommunikationspartner, deren WSDL-Beschreibungen sowie die Art und Weise (z.B. die Reihenfolge) wie Aufrufe zwischen den Partnern abgesetzt werden und wie die Gegenseite darauf reagiert. [CKMT$^+$03]

Die “Quality of Service”-Konzepte (QoS) wie *WS-Coordination* und *WS-Transaction* erlauben es Eigenschaften von Aufrufen zu definieren. Mit WS-Coordination können Koordinationsprotokolle zwischen Diensten definiert wer-

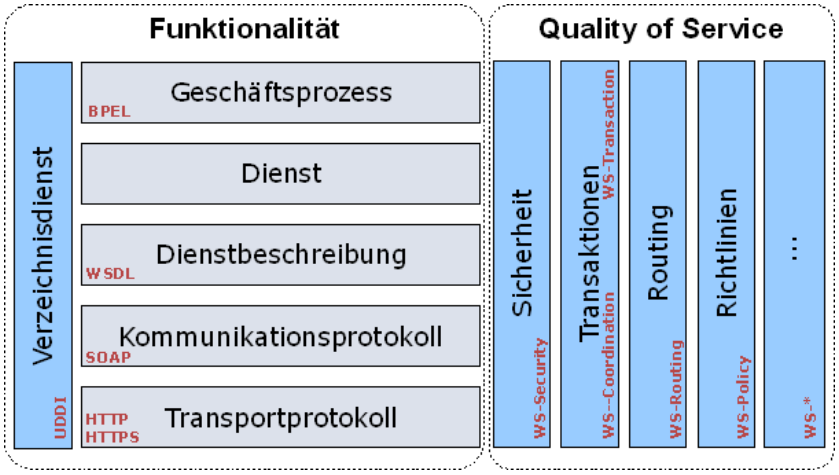


Abbildung 6. Web Services - Schichtenmodell (Vergleiche [EAAC+04])

den. Dies sind wohldefinierte Abfolgen von Nachrichten, die Dienste austauschen. Auf diese Weise können Benachrichtigungs- und Synchronisationsprotokolle definiert werden, die nach Bedarf abgewickelt werden können. WS-Transaction setzt auf dieser Idee auf und definiert zwei Koordinationsprotokolle, nämlich "Atomic Transaction" (AT) und "Business Transaction" (BT). Dabei sind ATs Transaktionen im herkömmlichen Sinne, die Aufrufe nach dem ACID-Prinzip garantieren. BTs hingegen setzen die Idee des "Unit of Work"-Musters [Fowl03] um. Dies sind in sich abgeschlossene Aktivitäten, die über einen langen Zeitraum durchgeführt werden können, wobei der Abbruch der Aktion jeder Zeit möglich ist. [CKMT+03]

WS-Security ist eine ebenfalls von OASIS vorgeschlagene Erweiterung des SOAP-Protokolls, die dazu verwendet werden kann Authentifizierungen und Zugangsbeschränkungen umzusetzen. Dazu wird im Kopfteil einer SOAP-Nachricht ein Definitionsblock namens "Security" angelegt, in dem Signaturen und andere Sicherheitsinformationen abgelegt werden, die von den Nachrichtenempfängern ausgewertet werden. [CHKT06],[ACKM04]

Es existieren mehrere weitere Konzepte um mit Richtlinien, Routing oder Statusverfolgung von Aufrufern umzugehen, auf deren Beschreibung hier verzichtet wird. [CHKT06], [ACKM04], [EAAC+04] liefern mehrere Übersichten und detaillierte Beschreibungen. Spezifikationen können unter `http://www.oasis-open.org` und `http://www.ws-i.org` eingesehen werden.

3.4 Enterprise Service Bus

Ein Enterprise Service Bus ist eine standardbasierte Integrationsplattform, die nachrichtenorientierte Kommunikation, Web Services, Datentransformation und

intelligentes Routing vereint. Sie wird dazu verwendet um Interaktionen zwischen einer Vielzahl unterschiedlicher Anwendungen innerhalb von Großunternehmen und über Unternehmensgrenzen hinaus verlässlich zu koordinieren. Als Service Bus (siehe Abschnitt 3.2) stellt er das Rückgrat einer dienstorientierten Architektur dar. [Chap04]

Die Grundidee des ESB ist es ein Kommunikationsnetz zu schaffen, in dem Anwendungen beliebig zugeschaltet werden können und damit die Funktionalität und Daten jeder anderen Anwendung, die an dem ESB hängt, nutzen können. Obwohl Web Services im ESB eine entscheidende Rolle spielen, ist es keine Voraussetzung, dass alle Anwendungen als Dienste zur Verfügung stehen. Vielmehr wird mit einer Reihe von Kommunikationsmechanismen und -protokollen, APIs und Adaptern gearbeitet, die auch die Integration von Altsystemen und Fertigsoftware ermöglichen (siehe Abbildung 7).

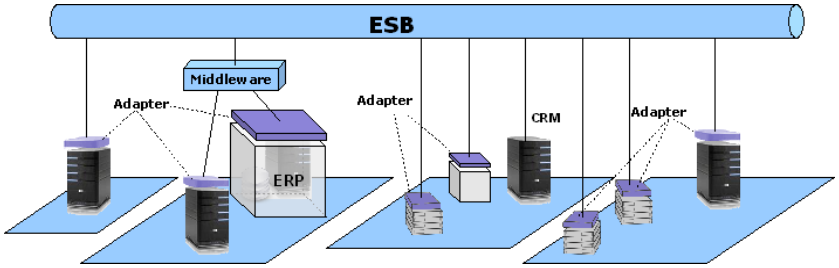


Abbildung 7. Enterprise Service Bus (Vergleiche [Chap04, S. 8])

Eine wichtige Eigenschaft des Enterprise Service Bus ist die Ausnutzung von Standards. Er basiert zum großen Teil auf Web Services und verwendet die im vorigen Abschnitt vorgestellten Standard-Web-Technologien. Darüber hinaus wird XML als Standardformat für die Kommunikation eingesetzt. Die weite Verbreitung von XML als Datenformat in Anwendungen und die Existenz ausgezeichneter Werkzeuge und XML-Technologien wie XSD (XML-Schema), XSLT (XML-Transformation), XPath (XML-Navigation) und XQuery (Anfragen auf XML-Dokumente) schaffen eine starke Basis für die Verarbeitung von Nachrichten.

Da in einer Anwendungslandschaft eines Großunternehmens eine starke Heterogenität in den verwendeten Datenformaten vorherrschen kann, ist es eine zentrale Aufgabe der EAI Nachrichtenformate in einander zu transformieren. Daher ist eine wichtige Anforderung des ESB Adapterdienste und Technologie-Gateways als Zwischendienste (siehe Abschnitt 3.2) zur Verfügung zu stellen. Hierfür werden zunächst für Nachrichten, die im Rahmen eines Geschäftsprozesses ausgetauscht werden, einheitliche XML-Formate festgelegt (Standardnachrichten). Bei der Verteilung von Nachrichten über den ESB werden dann ausschließlich diese Nachrichtenformate verwendet. Die Ein- und Ausgaben von An-

wendungen die über den ESB kommunizieren, müssen daher zunächst in Standardnachrichten umgewandelt werden. Dies kann über eine Zwischenschaltung eines Adapterdienstes, der die Formattransformation (ggf. mit XSLT) übernimmt, geschehen (siehe Abbildung 8). [Chap04]

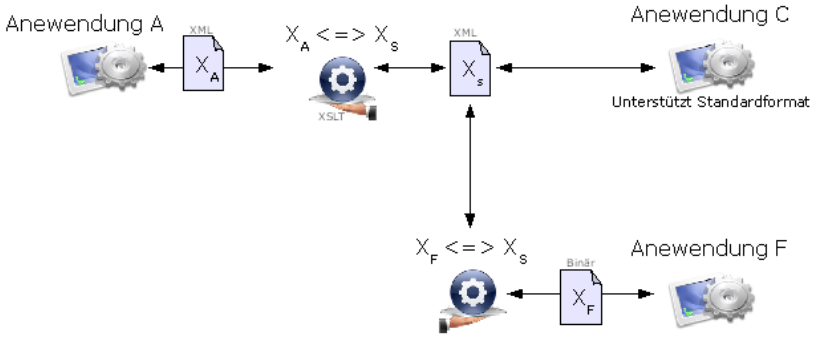


Abbildung 8. Nachrichtentransformation im ESB (Vergleiche [Chap04, S. 72ff])

Viele Konzepte der Middleware, unter anderem die MOM, (siehe Abschnitt 3.1) finden sich auch in der Umsetzung des ESB wieder. Statt jedoch eine zentrale Softwarekomponente zu verwenden, die für das Nachrichtenrouting und die -transformation eingesetzt wird, werden diese Funktionalitäten als Dienste an den Integrationspunkten zur Verfügung gestellt. Die Idee eines Busses wird hier in dem Sinne benutzt, dass jeder Teilnehmer, der am Bus angeschlossen ist mit jedem anderen kommunizieren kann. Dabei muss ein Teilnehmer nicht wissen *wie* die Kommunikation mit allen potenziellen Empfängern und Sendern funktioniert. Die einzigen Aufgaben, die er zu erfüllen hat, sind das Verbinden zum ESB, das Senden und das Empfangen von Nachrichten. Die Nachrichtenvermittlung des ESB, der als Kern eine verteilte nachrichtenorientierte Middleware zugrunde liegt, übernimmt eine verlässliche Zustellung an den oder die Adressaten. [Chap04]

Die Anbindung an den ESB wird in der Notation von “abstrakten Verbindungspunkten” modelliert. Hinter einem solchen Verbindungspunkt können sowohl Altsysteme als auch Fertigsysteme genauso wie Dienste liegen, unabhängig davon wie komplex oder einfach diese sind. Diese Betrachtungsweise erlaubt die Organisation der Kommunikation auf einer hohen Abstraktionsebene. ESB-Softwarehersteller liefern dem EAI-Architekten viele Werkzeuge, die ihn bei der Planung und Implementierung eines ESB unterstützen. Die Idee ist vor allem, dass um verfügbare Dienste zu verknüpfen nicht programmiert sondern nur konfiguriert werden muss.

Die abstrakten Verbindungspunkte werden als *Service-Container* umgesetzt, die im Grunde die Funktion des ESB selbst ausmachen. Dies sind separate Prozesse, die im Netzwerk verteilt ablaufen, Softwarekomponenten verwalten und diese durch eine Dienstschnittstelle im ESB vertreten. Dabei können einer oder mehrere Dienste in einem Service-Container zusammengefasst sein. Zum einen bieten Service-Container eine Abstraktion vom unterliegenden Nachrichtenprotokoll der Kommunikationsmiddleware. Nachrichten vom ESB werden im Standardformat an die angeschlossenen Dienste übergeben und von ihnen entgegengenommen. Eine verteilte nachrichtenorientierte Middleware dient als Kern für die Übertragung der Nachrichten zwischen den Service-Containern selbst und anderen (evtl. nicht dienstorientierten) Teilen der Anwendungslandschaft. Dabei kommen vor allem die Fähigkeiten der MOM zum Tragen, Nachrichten asynchron zu übermitteln sowie unterschiedliche Transportprotokolle und Übertragungsformate zu unterstützen.

Zum anderen wird durch das "Invocation and Management Framework" eine Vielzahl an Funktionalitäten bereitgestellt die bei der Implementierung und Anbindung von Diensten nützlich oder notwendig sind. Darunter fällt etwa die Unterstützung von Nachrichtennachverfolgung und Prüfung (Tracking & Auditing), Threading- und Transaktionsmanagement sowie Sicherheitsmechanismen und vieles mehr. (siehe Abbildung 9). [Chap04]

[Chap04]

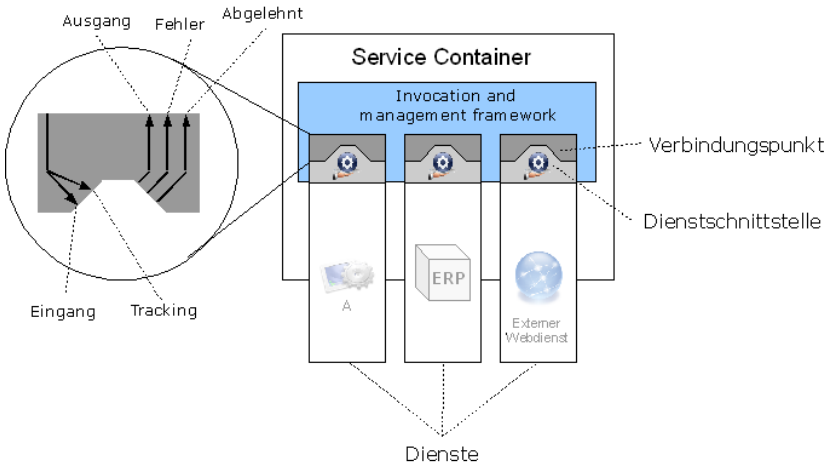


Abbildung 9. Drei Dienste in einem Service Container (Vergleiche [Chap04, S. 115])

Die Nutzung eines Dienstes über ein Service Repository in einer SOA benötigt die Implementierung einer Lokalisierung, der Bindung sowie der Aufrufe

der Schnittstellenmethoden. Jede Applikation, die einen Dienst benutzen möchte muss dies bewerkstelligen. Auch wenn die Kopplung des Dienstkonsumenten und des Dienstanbieters dynamisch zustande kommt, ist das Ergebnis stets eine Client-Server-Beziehung zwischen den Beteiligten. Ein Dienst ruft einen anderen auf. [Week06] Um Geschäftslogik auf Abfolgen von Dienstaufrufen abzubilden muss eine zusätzliche höhere Anwendungsschicht programmiert werden. Im ESB hingegen sind die Mechanismen zum Aufrufen von Diensten von der Logik getrennt. Es gibt keine direkten Aufrufe zwischen Diensten, stattdessen werden Nachrichten asynchron abgesetzt. Ein Dienst muss sich daher lediglich um die Implementierung kümmern und überlässt die Weiterleitung seiner Anfrage und gegebenenfalls einer Antwort des entsprechenden Zieldienstes oder anderen Instanz (z.B. einer Fehlermeldung eines Zwischendienstes) der Nachrichtenvermittlung. Die Verbindung der Dienste untereinander und die Aufstellung der Regeln für das Routing der Nachrichten eines Geschäftsprozesses passieren dabei nicht durch Programmlogik der Dienste, sonder durch eine werkzeuggestützte Konfiguration.

Durch die Ausnutzung der Vorteile von SOA und MOM-basierter Kommunikation bietet der Enterprise Service Bus eine sehr wertvolle Technologie für EAI. Seine verteilte Architektur erlaubt es auch sehr komplexe, physikalisch und organisatorisch getrennte Anwendungsinfrastrukturen zu integrieren. Die Bewältigung der Komplexität ist dabei ein sehr wichtiger Aspekt für die EAI. Die gesamte IT-Landschaft und notwendigerweise auch die damit verbundenen Unternehmensstrukturen zu begreifen, den Integrationsprozess zu planen und ihn umzusetzen ist eine extrem herausfordernde Aufgabe, die mehrere Jahre in Anspruch nehmen kann. Langzeitprojekte haben typischerweise mit sich immer wieder ändernden Anforderungen (Technologiewandel, Änderungen in der Organisationsstruktur) zu kämpfen, weswegen eine Einteilung der Phasen notwendig ist. [Reus06, Kap. 9]

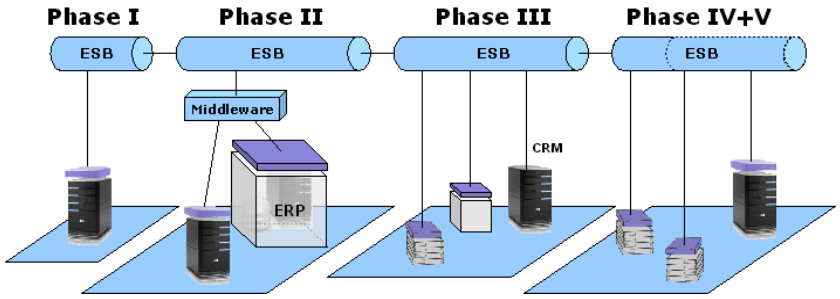


Abbildung 10. EAI-Projektphasen mit dem Enterprise Service Bus (Vergleiche [Chap04, S. 13])

Durch Interoperabilität mit unterschiedlichen Technologien und Formaten dank Middleware sowie dem Konzept der Service Container, liefert der ESB hier einen sehr guten Ansatz. Systeme werden schrittweise an den ESB angeschlossen (Abbildung 10). Bestehende Integrationsansätze, wie zum Beispiel RCP-basierte Punkt-zu-Punkt-Kommunikation, werden extrahiert und als "Brücken" in den ESB eingeklinkt. Diese Brückendienste simulieren die vorige Integrationsfunktionalität und kommunizieren mit der eigentlichen Anwendung über den Bus. Anwendungen außerhalb des eingeführten ESB bekommen davon nichts mit, was eine Integration in von einander unabhängigen Phasen ermöglicht. [Chap04]

Desweiteren bieten mehrere Softwarehersteller (z.B. IBM (WebSphere), Progress Sonic (Progress Sonic ESB)) und einige Open-Source-Projekte (OpenESB, Apache ServiceMix, JBossESB) unterschiedliche Implementierungen sowie Werkzeuge an, die den Aufbau und die Wartung eines ESB unterstützen. Darunter sind neben Implementierungen der Grundkonzepte (Service-Container, MOM) Hilfsmittel wie Dienstdesigner, graphische Umgebungen für Geschäftsprozessdefinition, und viele mehr. [CHKT06][Wiki08a]

3.5 Open SOA - SCA und SDO

Relativ neue Vorschläge für den Umgang mit dienstorientierten Architekturen sind *Service Component Architecture*(SCA) und *Service Data Objects* (SDO), die von der "Open Service Oriented Architecture Collaboration" (OSOA) spezifiziert wurden.

SCA bietet ein Programmiermodell für eine dienstbasierte Architektur an, das auf dem Konzept von Komponenten (siehe [Reus06, Kap. 4 und 5]) basiert. Die SCA-Spezifikation [OSOA07a] führt eine neue Notation für die Beschreibung von Komponenten (Component), Komponentenkomposition (Composite), sowie der Interaktionseigenschaften (Services & References) zwischen Komponenten innerhalb und außerhalb der Laufzeitumgebung (Domain), in der sie ausgeführt werden, ein.

Komponenten werden als die Bausteine der Umsetzung von einzelnen Geschäftsfunktionen verstanden, die bestimmte Konfigurationsmöglichkeiten (*Properties*) zur Verfügung stellen. Zusätzlich wird definiert welche Dienste eine Komponente bereitstellt (Services), was etwa durch WSDL definiert werden könnte, und von welchen Diensten die Komponente abhängt (References). Durch sogenannte Wires (innerhalb von Komponenten) und Bindings (zwischen Komponenten) kann dann definiert werden welche Komponenten mit welchen zusammenarbeiten. Auf diese Weise entstehen Kompositionen von Komponenten (Composites), die wiederum als Schnittstellen References und Services definieren und wiederum zu Kompositionen zusammengeschaltet werden können.

Als weiteres Konzept werden Laufzeitumgebungen (Domains/Domänen) als abstrakte Systemgrenze für eine Menge von Komponenten und Kompositionen eingeführt. Während an die Verbindungsarten zwischen Komponenten innerhalb von Domänen keine Anforderungen gestellt werden, wird verlangt, dass

die Kommunikation über Domänengrenzen hinaus (also andere Domänen, oder nicht-SCA-Anwendungen) über Web-Dienste abgewickelt wird. Abbildung 11 zeigt eine schematische Darstellung einer Verteilten SCA-Anwendung.

Ziel ist es durch Verwendung von SCA wiederverwendbare Softwarekomponenten und komponentenbasierte Anwendungen herzustellen. Die Geschäftslogik, die eine Komponente umsetzt, wird von den Verbindungsspezifika der Dienste (Lokalisierung, Bindung und Aufrufen) getrennt. Diese werden zusammen mit Anforderungen an die Infrastruktur und Kommunikation wie Richtlinien, Transaktion, Sicherheit, etc. (siehe Abschnitt 3.3) durch eine Konfiguration beschrieben. [OSOA07a][Chap07]

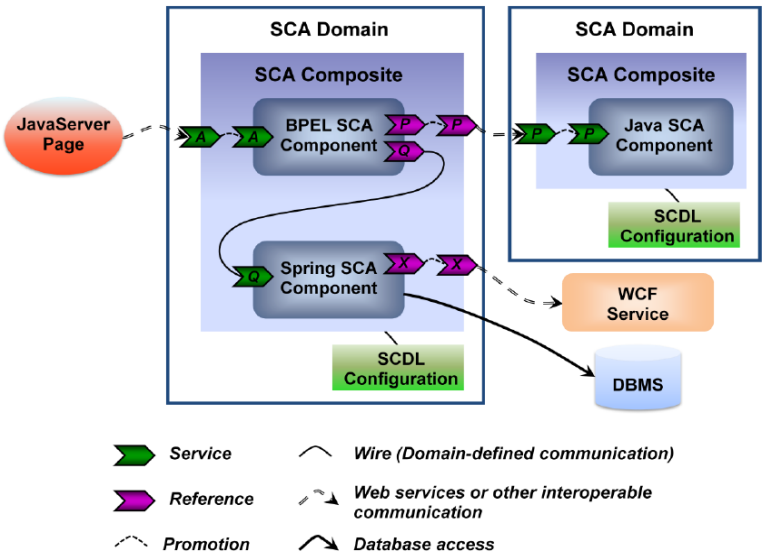


Abbildung 11. Beispiel einer SCA-Anwendung (Quelle [Chap07, S. 19])

Die SCA stellt ein flexibles und erweiterbares Modell für die komponentenbasierte Softwareentwicklung dar und eignet sich auch für die Beschreibung von Enterprise-Applikationen oder Teilen von EAI-Lösungen. Weil es aber eben nur ein Modell ist - eine abstrakte Beschreibung von Anwendungsstrukturen - liefert es zumindest keine direkten Lösungsansätze für die EAI.

SDO spezifiziert ein vereinheitlichtes Datenzugriffsmodell. Es ermöglicht eine einfache Behandlung von Daten aus unterschiedlichen Datenquellen, in dem es eine höhere Schicht der Datenabstraktion einführt. SDO unterstützt die Umsetzung von Entwurfsmustern für Abstraktionen der Datenzugriffsschicht und stellt

sowohl eine statische als auch eine dynamische Datenzugriffsschnittstelle auf die Datenobjekte zur Verfügung. Im statischen Fall wird eine Abbildung des Datenschemas auf eine Schnittstelle verwendet und im dynamischen Fall kann auf die Daten per "Datenmengen" zugegriffen werden (ResultSet in JDBC, DataSet in .NET sind Beispiele für Datenmengen).

Neben den Data Objects sind in SDO sogenannte *Datengraphen* (*Data Graphs*) ein zentrales Konzept. Während Data Objects die eigentlichen Daten entsprechend dem zugrunde liegenden Datenschema halten, ist durch die Datengraphen eine Datenstruktur umgesetzt worden, die die Datenobjekte und alle Veränderungen (Change Summaries) auf diesen verwalten. Datenzugriffsdienste (Data Access Services) haben die Aufgabe mit der Datenquelle zu kommunizieren und Datengraphen mit dem Datenbestand zu synchronisieren. Dabei gilt als zusätzliche Anforderung, dass ein Datengraph von der Datenquelle abgetrennt ist und eine Verbindung nur für Datesynchronisationen aufbaut. Abbildung 12 zeigt die grundlegende Struktur der SDO-Komponenten. [OSOA06] [OSOA07b]

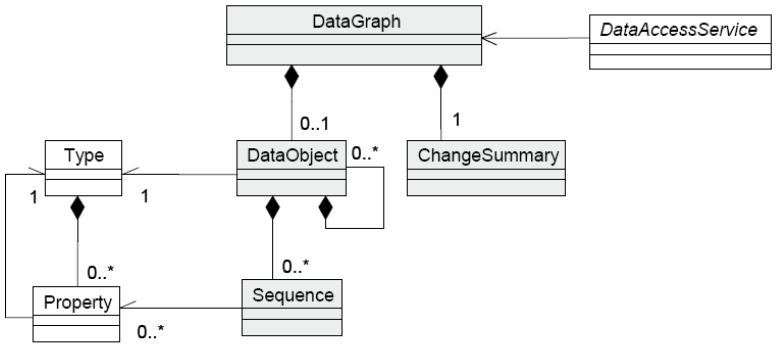


Abbildung 12. UML-Modell der grundlegenden SDO-Komponenten (Quelle [OSOA07b, S. 7])

Die Vorteile eines einheitlichen Datenzugriffmodels liegen auf der Hand. Die Behandlung unterschiedlicher Datenquellentypen und Datenquellen kann entfallen. In der Entwicklung von Anwendungen werden Daten bereits als Objekte im Sinne der objektorientierten Programmierung geliefert und erlauben es Architekten und Entwicklern sich auf die Umsetzung der Applikationslogik zu konzentrieren. Service Data Objects eignen sich somit im Kontext von dienstorientierten Architekturen für eine Integration auf Datenebene.

4 Zusammenfassung

Obwohl das Thema der Enterprise Application Integration seit mehreren Jahrzehnten existiert und heutzutage mächtige Technologien und unterstützende Werkzeuge für dieses Gebiet verfügbar sind, bleibt EAI für jedes Großunternehmen eine Herausforderung. Wir haben versucht einen Einblick in die EAI zu geben und zu erklären welche Umstände den Einsatz von EAI-Technologien fordern und fördern. Einige Integrationskonzepte, die teilweise seit den Anfängen der 60er Jahren bekannt sind, kommen noch heute zum Einsatz. Bis hin zu den aktuell sehr populären dienstorientierten Architekturen (SOA) werden alle Strategien - wenn auch auf unterschiedlichen Ebenen - verwendet. Der Enterprise Service Bus zusammen mit Konzepten der Web Services, SCA und SDO ist dabei ein möglicher Ansatz einer fortgeschrittenen EAI, die sich voll auf dienstorientierte Konzepte und Standardtechnologien stützt und damit auch für die Zukunft relevant bleibt.

Literatur

ACKM04. Gustavo Alonso, Fabio Casati, Harumi Kuno und Vijay Machiraju. *Web services : concepts, architectures and applications.* Data-centric systems and applications. Springer, Berlin. 2004.

BrJK03. Alan Brown, Simon Johnston und Kevin Kelly. Using service-oriented architecture and component-based development to build Web service applications, Nov 2003. IBM.

Chap04. David A. Chappell. *Enterprise service bus.* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol,CA 95472, USA. 2004.

Chap07. David Chappell. Introducing SCA. Technischer Bericht, David Chappell and Associates, OSOA, 2007.

CHKT06. Stefan Conrad, Wilhelm Hasselbring, Arne Koschel und Roland Tritsch. *Enterprise Application Integration : Grundlagen - Konzepte - Entwurfsmuster - Praxisbeispiele.* Elsevier, Spektrum Akad. Verl., Heidelberg. 1. Aufl.. Auflage, 2006.

CKMT+03. Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai und Sanjiva Weerawarana. The next step in Web services. *Commun. ACM*, 46(10), 2003, S. 29–34.

EAAC+04. Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, PO-El Krogdahl, Dr Min Luo und Tony Newling. *Patterns: Service-Oriented Architecture and Web Services.* IBM Redbooks. 2004.

Fowl03. Martin Fowler. *Patterns of enterprise application architecture.* Addison-Wesley. 2003.

Gras05. Computerwoche Stefan Grasmann. SOA-Risiken werden gern verschwiegen, 2005. [Online; accessed 04-July-2008].

Kell02. Wolfgang Keller. *Enterprise Application Integration : Erfahrungen aus der Praxis.* dpunkt-Verl., Heidelberg. 1. Aufl.. Auflage, 2002.

KrBS04. Dirk Krafzig, Karl Banke und Dirk Slama. *Enterprise SOA : Service-Oriented Architecture Best Practices (The Coad Series).* Prentice Hall PTR. November 2004.

Lint00. David S. Linthicum. *Enterprise application integration.* Addison-Wesley Longman Ltd., Essex, UK. 2000.

Lint03. David S. Linthicum. *Next Generation Application Integration: From Simple Information to Web Services.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 2003.

OSOA06. Open Service Oriented Architecture Collaboration OSOA. SDO - Service Data Objects For Java Specification. Technischer Bericht, OSOA, 2006.

OSOA07a. Open Service Oriented Architecture Collaboration OSOA. SCA - Service Component Architecture, Assembly Model Specification. Technischer Bericht, OSOA, 2007.

OSOA07b. Open Service Oriented Architecture Collaboration OSOA. Service Data Objects White Paper. Technischer Bericht, Open Service Oriented Architecture Collaboration, 2007.

PaHe07. Mike P. Papazoglou und Willem-Jan Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3), 2007, S. 389–415.

Reus06. Ralf H. [Hrsg.] Reussner (Hrsg.). *Handbuch der Software-Architektur.* dpunkt-Verl., Heidelberg. 1. Aufl.. Auflage, 2006.

Week06. Computerwoche Eerko Weeke. Die Spaghetti-Falle droht auch bei SOA, 2006. [Online; accessed 04-July-2008].

Wiki08a. Wikipedia. Java Business Integration, 2008. [Online; accessed 05-June-2008].

Wiki08b. Wikipedia. Transaktionsmonitor, 2008. [Online; accessed 04-June-2008].

Service Orientierte Architektur (SOA)

Jaouad Bouras

Betreuer: Dipl.-Inform. Henning Groenda

Zusammenfassung Die Service Orientierte Architektur (SOA) ist ein neues *abstraktes Konzept* im Bereich der *Informationstechnologie*. Die Hauptziele einer solchen Architektur ist das Anbieten, Suchen und Nutzen von so genannten *Diensten*. In einem Netzwerk interagieren die Anwendungen und Applikationen miteinander, in dem sie gegenseitige Dienste nutzen. Anhand von einheitlichen Standards werden Dienste und ihre Schnittstellen beschrieben, was den Vorteil hat, dass von der zugrundeliegenden Hardware- und Softwareschicht der jeweiligen Applikationen abstrahiert werden kann. Im Bereich der Geschäftsprozessen und ihre Modellierung konkurrieren die zwei weit verbreitete Sprachen XPDL und BPEL. Diese werden wir in diesem Paper genauer untersuchen und vergleichen.

1 Service Orientierte Architektur

1.1 Definition

Zurzeit existiert keine genaue Definition des Begriffs SOA. Dies liegt nicht daran, daß es keine Definition gibt, sondern, dass SOA einerseits eine neue Technologie ist und andererseits, von mehreren Standards unterstützt ist. Die folgenden Definitionen unterscheiden sich im Kontext, Grad der Abstraktion und in der Formulierung.

In [WDB.05] wird von den wesentlichen Merkmalen einer SOA ausgegangen, um eine Definition abzuleiten.
Diese Merkmale sind:

- **Lose Kopplung:** Unter diesem Merkmal versteht man das dynamische Suchen, Finden und Einbinden von Diensten durch Anwendungen oder andere Dienste. Außerdem bedeutet die lose Kopplung, dass jede Software-Einheit unabhängig von einer anderen ist. Dies hat den Vorteil, dass eine Änderung in einer Software-Komponente weniger Aufwand und Kosten verursacht, als in einem System mit starken Abhängigkeiten.

- **Dynamisches Binden:** Dieses charakterisiert das binden von Diensten zur Laufzeit als auch außerhalb der Anwendungprogramme, dadurch kann während der Entwicklungszeit von den konkreten Informationen des aufrufenden Dienstes abstrahiert werden.

– **Verzeichnisdienst:** Da Dienste dynamisch als auch statissh gesucht werden, müssen sie an einem Ort bekanntgemacht und registriert werden. Als Beispiel stellt das **UDDI**-Verzeichnis (Universal Description, Discovery and Integration Registry) Schnittstellen zum Suchen von Diensten zur Verfügung und kann Rechte für die Zugriffe sichern.

– **Offene Standards:** Durch die Verwendung von offenen Standards können bestehende als auch neue Anwendungen in Form von Diensten bereitgestellt und integriert werden. Dadurch definiert SOA ein Framework anstatt eine Unternehmen-IT auf der Basis der Service-Idee neu realisieren.

Aus diesen Merkmalen lässt sich laut [WDB.05] folgende Definition herleiten „Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht“.
Die *Organization for the Advancement of Structured Information Standards* (OASIS) ist eine internationale, nicht-gewinnorientierte Organisation, die sich mit der Weiterentwicklung von E-Business- und Web-Service-Standards beschäftigt [srcs06]. Sie hat SOA wie folgt definiert: „SOA ist ein Paradigma für die Strukturierung und Nutzung verteilter Funktionalität die von unterschiedlichen Besitzern verantwortet wird.“. Das *W3C*, das Gremium zur Standardisierung der das World Wide Web betreffenden Techniken, hingegen definiert SOA in [Note04] als Form einer auf Web Services basierten verteilten Anwendung, in der mehrere Agenten über ein Netzwerk interagieren.

1.2 Realisierung

Die einfache Realisierung von SOA erfordert keinen großen Aufwand und muß kein unhandliches IT-Projekt sein. Als erstes sollen die grundlegenden Dienste ihre Funktionen richtig erfüllen und mit einfachen Mitteln betrieben werden. Im Unternehmen sollen klare Anforderungen und vollständige Vorstellungen der Strategien definiert werden. Denn anfängliche Fehler könnten im Nachhinein in eine im Laufe der Zeit ansteigende kostspielige Korrektur resultieren.
Die Infrastruktur ist der technische Teil einer SOA um Interoperabilität zu ermöglichen und wird oft als *ESB* (Enterprise Service Bus) bezeichnet. Die zentrale Aufgabe eines ESB ist der Austausch von Daten, in Form von Dienst-Aufrufen, zwischen IT-Systemen oder Teilen von IT-Systemen. Zu seinen Aufgaben gehören auch die Datentransformation, Vermittlung, Betriebssicherheit, Dienstver-

waltung, Überwachung und Protokollierung [Josu07].

1.3 Zusammenfassung

In den verschiedenen Definitionen ist vereinigt, dass SOA ein Paradigma für verbesserte Flexibilität ist. In der Objektorientierung stellen Anwendungen ihre Funktionalitäten durch wohldefinierte Schnittstellen zur Verfügung, und können dadurch ihre Details für die aufrufende Applikation verbergen. Dadurch lässt sich der Objektorientierten Ansatz als eine mächtige Programmierparadigma bezeichnen. Allerdings muss in verteilten Systemen, wo verschiedene Plattformen mit einander kommunizieren, die aufrufende Anwendung in derselben Sprache, wie das Objekt auf das zugegriffen wird, geschrieben werden. In SOA hingegen bieten z.B. *Web Services*, Dienste, die für die Kommunikation zwischen Computern konzipiert und über verschiedene Plattformen hinweg interoperabel sind, eine standarisierte Art der Kommunikation zwischen Dienste, die in beliebigen Sprachen geschrieben werden können.

2 Orchestration und Choreographie

2.1 Services

Ein Dienst engl. „*Service*“ ist ein zentraler Punkt in der Service Orientierten Architektur, er existiert als ein physikalisches eigenständiges Softwareprogramm. Einerseits sind Dienste mit eindeutigen Entwurfseigenschaften charakterisiert und können als Module betrachten werden. Andererseits können sich Dienste auch komponiert werden, um große Leistung beizubringen. Auch wenn jeder Dienst als individuell betrachtet ist, muss für das effektive Funktionieren in größeren Unternehmen grundlegende nicht funktionale Eigenschaften wie Verfügbarkeit, Zuverlässigkeit und die Fähigkeit der Zusammenarbeit mit anderen Komponenten umgesetzt werden. Das Erfüllen dieser Art von Anforderungen ist das zentrale Ziel der Serviceorientierung.

2.2 Web Services

Häufig werden *Web Services*[Newc05] in der Realisierung einer SOA eingesetzt. Sie dienen zur Bereitstellung von Funktionalitäten auf der Anwendungs- order Geschäftsebene. Mit Hilfe von standardisierten Schnittstellen können Web Services in einer einfachen Form über Internet-Protokolle aufgerufen werden.
Die Technologie von Web Services basiert auf der Kombination mittels *XML/SOAP* (Simple Object Access Protocol) über HTTP. Ein Web Service-Protokoll besteht aus mehreren Schichten, die für die Implementierung einer SOA notwendig sind. Dieses Bündel wird auch „*Web Service Stack*“ genannt und ist in Abbildung 1 graphisch dargestellt.

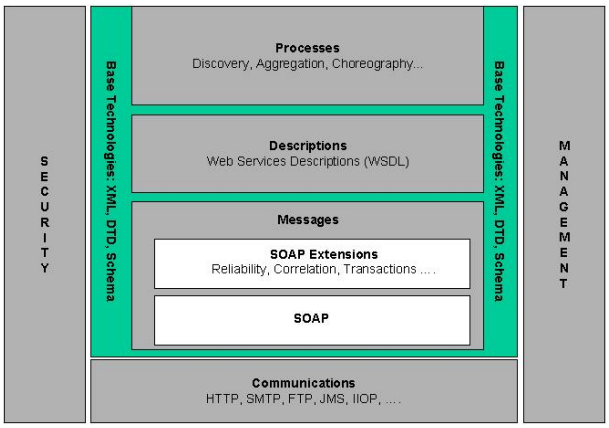


Abbildung 1. Web Service Stack (Quelle : [W3C])

Um die wichtigsten Teile eines Web Services, den Dienstanbieter, das Dienstverzeichnis und die Dienstnutzer sowie ihren Zusammenhang zu verstehen, werden diese Anhand eines Beispiels erklärt. In der realen Welt kann die Suche nach einem Dienst z.B. einer Computerreparatur folgendermaßen verlaufen: Als erstes suchen Sie als Dienstnutzer in einem Dienstverzeichnis z.B. den gelben Seiten anhand der im Dienstverzeichniss vorhandenen Informationen (in diesem Fall Adresse und Telefonnummer). Sie kontaktieren den Dienstanbieter, der den gesuchten Dienst erbringt. Dieser Zusammenhang ist in Abbildung 2 veranschaulicht. Schrittweise gesehen veröffentlicht als erstes der Dienstanbieter Geschäftsinformationen im Dienstverzeichnis. Danach sucht der Dienstnutzer nach einem Dienst, der Dienstanbieter wird aufgerufen und mit ihm kommuniziert um den Dienst auszuführen.

Auf technischer Ebene werden anhand von SOAP und anderen Netzwerkprotokollen Nachrichten im Netzwerk transportiert. Die SOAP-Spezifikation [WRNi03] legt ausschließlich fest, wie eine Nachricht aufgebaut werden muss, um als SOAP-Nachricht gelten zu können. Die Kommunikation zwischen Web Services mittels SOAP ist plattformunabhängig und ist nicht an eine bestimmte Programmiersprache bzw. ein bestimmtes Betriebssystem gebunden.

Ein anderer Bestandteil in der Beschreibung von Web Services ist die *WSDL* (Web Service Description Language). WSDL [W3C12] ist eine XML-basierte Beschreibungssprache für Web Services Schnittstellen. Die WSDL behandelt eine abstrakte und eine konkrete Ebene. Hierbei wird zwischen von der Beschreibung der Funktionalität eines Web Services und den technischen Eigenschaften wie z.B. dem Ort und der Art und Weise, wie der Dienst angeboten wird, getrennt.

Die dritte grundlegende Komponente ist das *UDDI-Protokoll* (Universal Des-

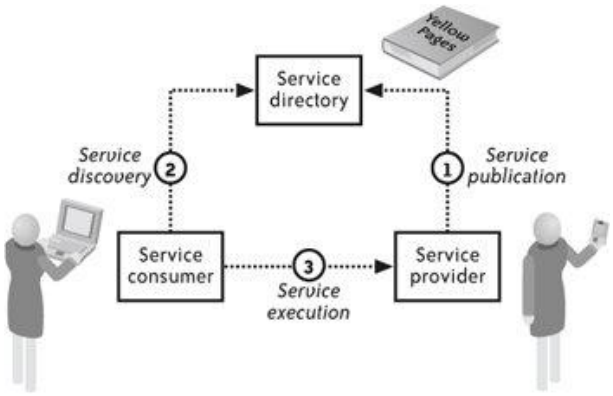


Abbildung 2. Web Service Verlauf (Quelle : [PeRD00])

cription, Discovery and Integration protocol). Es beschreibt einen Verzeichnisdienst für Web Services und spezifiziert eine standarisierte Verzeichnisstruktur für die Verwaltung von Web-Service-Metadaten. Zu den Metadaten gehören auch allgemeine Anforderungen, Web-Services-Eigenschaften und die benötigten Informationen zum Auffinden von Web Services.

2.3 Unterschiede zwischen Orchestration und Choreographie

Die Begriffe Orchestration und Choreographie beschreiben zwei Aspekte für die Erstellung von Geschäftsprozessen aus zusammengesetzte Web Services. Die beiden Begriffe werden oft falsch benutzt und als Synonyme verwendet. Die Orchestration bezieht sich auf einen ausführbaren Geschäftsprozess, der mit anderen internen sowie auch externen Web Services interagiert. Dieser zentrale Prozess übernimmt die Kontrolle der Steuerung zwischen den beteiligten Diensten und koordiniert die Ausführung der verschiedenen Operationen zwischen den teilnehmenden Diensten. Diese brauchen nicht zu wissen, dass sie Teil der Komposition einer Operation oder eines großen Geschäftsprozesses sind.
Die Choreographie hingegen verwendet keinen zentralen Koordinator. Jeder beteiligte Dienst weiß genau wann er welche Operation auszuführen hat und mit wem er interagieren und Kontakt aufnehmen soll. Choreographie ist somit ein kollaborativer Einsatz basierend auf dem Austausch von Nachrichten. Alle Teilnehmer einer Choreographie sollen sich über den Teilsprozess, die Operationen die sie ausführen, die Nachrichten die sie austauschen müssen, und den zeitlichen Ablauf bewusst sein.

Wie die Abbildung 3 zeigt, ist die Choreographie kollaborativer Prozess und erlaubt jeder der beteiligten Komponenten sich zu beschreiben. Während die Orchestration sich nur auf einen spezifischen Prozess, der eine einzelne Komponenten ausführt beschränkt, verfolgt die Choreographie die Nachrichten zwischen verschiedenen Komponenten und Quellen. Dies ist typischerweise ein direkter Nachrichtenaustausch zwischen Web Services [Pelt03].

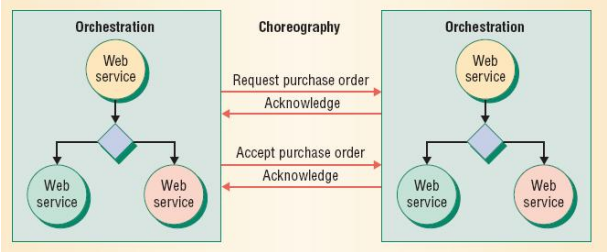


Abbildung 3. Orchestration vs Choreographie (Quelle : [Pelt03])

2.4 Business Process Management

In SOA sind Dienste Teile von einem oder mehreren Geschäftsprozessen. Dieser Zusammenhang stellt die Frage wie Dienste identifiziert werden sollen und führt zu dem wichtigen Begriff des Geschäftsprozessmanagements.
In der Literatur trifft man in diesem Zusammenhang oft auf das Akronym *BPM* welcher aber zwei unterschiedliche Bedeutungen besitzt. Einerseits steht es für das Geschäftsprozessmanagement also für die Verwaltung und Verbesserung von Geschäftsprozessen. Andererseits kann BPM auch für die Geschäftsprozessmodellierung (Business Process Modelling) verwendet werden. Im Allgemeinen ist die Geschäftsprozessmodellierung Teil des Geschäftsprozessmanagements.
Eine andere Verwirrung ist der Unterschied zwischen Geschäftsprozessmanagement und Workflowmanagement. Beide unterstützen hauptsächlich eine serviceorientierte Sicht in Systemen und Organisationen, und beide Begriffe werden meist als Synonym verwendet. Sie unterscheiden sich in der Form, wie ein Geschäftsprozess beschrieben wird. Um den Unterschied genauer zu verstehen, betrachten wir einen Geschäftsprozess aus allgemeiner Sicht.
[Josu07] definiert diese Begriffe folgendermaßen:

- „Ein Geschäftsprozess beschreibt *was* gemacht wird (inklusive Eingabe und Ausgabe). Er beschreibt welche Zeile zu erreichen sind und welche Vorraussetzungen zu erfüllen sind.
- Ein Workflow beschreibt *wie* ein bestimmtes Ergebnis erreicht wird. Er betrachtet die Details aller Schritten und Aktivitäten.“

In Abbildung 4 wird dieses Workflow Konzept veranschaulicht und genauer betrachtet. Der in Abbildung 4 dargestellte Geschäftsprozess (*Business process*) besteht aus mehreren Teilprozesse (*Subprocess*), die wiederum aus anderen Teilprozessen bestehen können. Jeder Teilprozess lässt sich dann in einzelne Aktivitäten (*Activity*) bestehend aus atomaren Schritten (*Step*) zerteilen. Die zwei letzten Komponenten eines Geschäftsprozesses gehören somit zur Workflowschicht.

Herstellern von Workflow-Management-Systemen stellen zu ihren Produkten Referenzmodelle zu Verfügung. Diese beschreiben wie das angebotene System Geschäftsprozesse unterstützt. Heute gibt es eine Reihe von Sprachen zur

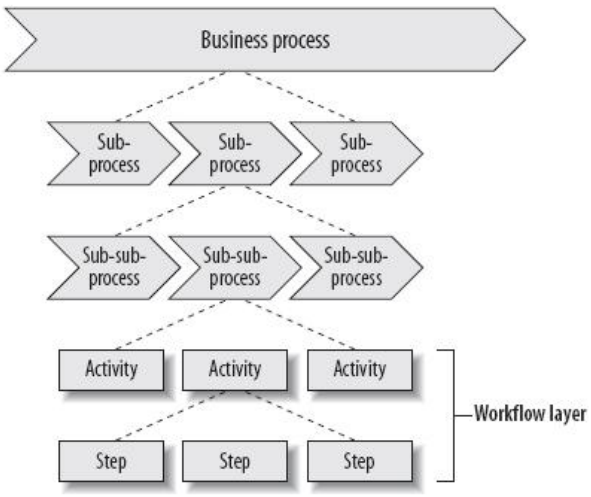


Abbildung 4. Hierarchie eines Geschäftsprozess mit einer Workflowschicht (Quelle : [Josu07])

Spezifikation von Prozessmodelle und Workflows, davon gibt es EPCs (Enterprise Process Centers), BPML (Business Process Modeling Language), Protos, SAP WebFlow, Yawl[HHAM05] und andere. Viele Workflowsprachen werden oft mit Petrinetzen oder Graphen modelliert und werden im Laufe der Zeit optimiert[GAPJV+08] und verbessert.

2.5 Business Process Execution Language

Definition. BPEL (Business Process Execution Language) spielt eine wichtige Rolle in der Modellierung und Ausführung von Geschäftsprozessen auf Basis von Web Services in Tools und Engines. BPEL ist einer der Standards für die Gestaltung und Ausführung von Geschäftsprozessen in SOAs geworden. BPEL wurde von Microsoft, IBM und BEA entwickelt. Sie vereinigt zwei Vorgänger XLANG und WSFL (Web Service Flow Language) und wurde der OASIS, dank der Unterstützung von einem Konsortium aus über 130 Firmen, im Jahr 2004 zur Standardisierung übergeben.

BPEL als Programmiersprache. Die Sprache BPEL hat eine XML-Grammatik für die Beschreibung der notwendigen logischen Steuerung und Koordination der beteiligten Web Services. Das vom Geschäftsprozess-Modellierer erstellte BPEL-Dokument dient als Eingabe für einen Prozess-Manager oder eine Workflow-Engine, welche diese Anweisungen ausführen, die Aktivitäten, sowie den gesamten Prozess koordinieren und mögliche Fehlerfälle behandeln. In welcher Weise sie dies letztendlich umsetzen und auf welcher Plattform sie entwickelt wurden ist nicht festgelegt und muss von den Entwicklern der Applikation auch

nicht betrachtet werden. Zwingend ist nur, dass die Workflow-Engine bzw. der Prozess-Manager die Anweisungen aus den BPEL-Dokumenten der Spezifikation [OASI07] ausführen kann.
BPEL verfügt über zwei Programmiermodelle (Abbildung 5):

- **Abstract business process model:** Sie spezifiziert den öffentlichen Nachrichtenaustausch zwischen den beteiligten Diensten. Die Geschäftsprotokolle sind nicht ausführbar und beinhalten keine Details über den internen Ablauf des Prozesses. Möchten Unternehmen ihre Prozesse verknüpfen, so tauschen sie abstrakte Prozessbeschreibungen aus, wodurch jedes Unternehmen die für die Interaktion notwendigen Informationen bekommt, aber nicht seine internen Abläufe verrät.

- **Executable business process model:** Sie modelliert das Verhalten der Beteiligten einer spezifischen Geschäftsinteraktion. Ausführbare Prozesse beinhalten Informationen über Schnittstellen, Variablen, Nachrichtenformate, Prozesslogik und andere. Sie enthalten aber keine Informationen über die Endpunkte eines Web Services, wodurch sich ein ausführbarer Prozess unabhängig von seiner späteren Ausführungsumgebung modellieren lässt. Mit diesem Programmiermodell wird dann die Orchestrierung von Web Services umgesetzt.

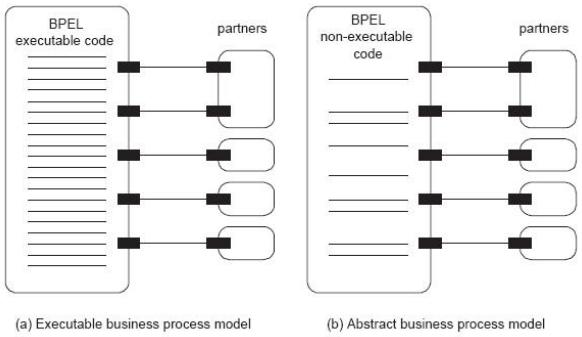


Abbildung 5. Die zwei Programmiermodelle von BPEL (Quelle : [ADHR+05])

Basisstruktur und Gültigkeitsbereiche. Wie in der Definition der Sprache BPEL angedeutet, wird jede Aktivität eines Geschäftsprozesses genauer beschrieben. In einem BPEL-Dokument wird immer die Wurzel mit dem Tag "*process*" identifiziert. In der Wurzel muss mindestens eine Aktivität enthalten sein (siehe Listing 2.1).

Listing 2.1. Struktur der ersten Ebene in einem BPEL-Dokument

```
<process name=''Name des Prozesses''>
  <partnerLinks>..</partnerLinks>
  <variables>..</variable>
  <correlationSets>..</correlationSets>
  <faultHandlers>..</faultHandlers>
  <componsationHandlers>..</componsationHandlers>
  <eventHandlers>..</eventHandlers>

  <!-- Hier mindestens eine Aktivität erwähnen -->
  <!-- Hier findet der Prozessablauf statt -->
</process>
```

Wie in der Programmiersprache Java existieren die Variablen in einem definierten Gültigkeitsbereich. In der Sprache BPEL wird dieser Bereich durch das Element "*scope*" definiert. Auch in BPEL gilt die Verdeckungsregel, dass lokale Variablen die globalen Variablen überlagern (Listing 2.2).

Listing 2.2. Definition von Gültigkeitsbereichen in einem BPEL-Dokument

```
<process name=''Name des Prozesses''>
  <!-- Äußerster Gültigkeitsbereich -->

  <scope>
    <!--  Aktivitäten und/oder neuer Gültigkeitsbereich-->
  </scope>

  <!--  Aktivitäten und/oder neuer Gültigkeitsbereich-->

</process>
```

Daten. Mit Hilfe von BPEL werden verschiedene Arten von Daten durch Variablen definiert. Diese können sowohl Daten zwischen Prozess und externen Partnern als auch Werte, die während der internen Auswertung der Geschäftslogik benötigt werden darstellen. In BPEL können Variablen als *WSDL message*, *XML Schema simple type* oder als *XML Schema element* deklariert werden [W3C12]. Listing 2.3 veranschaulicht die Deklaration von Variablen mit verschiedenen Datentypen. Mithilfe des Tags "*assign*" können Werte nicht nur von fixen Werten sondern auch von anderen Variablen zugewiesen werden. Mit der Manipulation der Werte von Variablen im Rahmen des Kontrollflusses eines Prozesses können verschiedene Ausdrücke aufgebaut und Regeln ausgewertet werden. BPEL unterstützt in diesem Kontext Ausdrücke mit Wahrheitswerten (Boolean Expressions), Ausdrücke mit Stichtagswerten (Deadline-Valued Expressions), Ausdrücke mit Zeiträumen (Duration-Valued Expressions) und allgemeine Ausdrücke (General Expressions). Die Formulierung von Ausdrücken wird mithilfe des Attributes *expressionLanguage* definiert.

Listing 2.3. Variablen in einem BPEL-Dokument

```
<process name=''Name des Prozesses''>
  <variables>
    <variable name="zahl" type="xsd:int"/>
    <variable name="strasse" messageType=""/>
    <variable name="bezeichner" element=""/>
  </variales>
</process>
```

Aktivitäten. In BPEL werden die einzelnen Arbeitsschritte eines Geschäftsprozesses durch Aktivitäten abgebildet. Man unterscheidet zwischen *elementaren* und *strukturierten* Aktivitäten. Jede Aktivität wird durch ein separates Element in einem BPEL-Dokument repräsentiert, in dem folgende Attribute definieren können (siehe auch Listing 2.4):

- **name** Mit diesem Attribut können Aktivitäten mit eindeutigen Bezeichnern definiert werden. Dies kommt zum Tragen, wenn eine Aktivität an verschiedenen Stellen des Prozesses verwendet werden soll. Die Wiederverwendung erfolgt dann über dieses optionale eindeutige Attribut.

- **joinCondition** Dies ist auch ein optionales Attribut und dient zur Ausführung einer Aktivität in Abhängigkeit zu einer oder mehr Bedingungen.

- **suppressJoinFailure** Dieses Attribut, mit dem Rückgabewert *true* (bzw. wahr) oder *false* (bzw. falsch), zeigt ob beim Kombinieren zweier Ausführungszweige einen Fehler aufgetreten ist und ob dieser unterdrückt werden soll.

Listing 2.4. Attribute einer Aktivität in BPEL

```
<activity
  name=''aktivität''
  joinCondition=''true''
  suppressJoinFailue=''true''
  ...
  >

</activity>
```

Zu den elementaren Aktivitäten gehören:

- **invoke** Da das Ziel von BPEL die Interaktion mit anderen Web Services ist, werden durch die Aktivität *invoke* nicht nur asynchrone Aufrufe sondern auch synchrone Request-/Response-Aufrufzyklen ausgeführt. Während bei asynchronen Aufrufen die Angabe einer *inputVariable* ausreicht, ist bei

synchronen Aufrufzyklen die Angabe einer zusätzlichen *outputVariable* zwingend notwendig. Die Synchronität der Aufrufzyklen ist in verteilten Systemen von großer Bedeutung, da in den Interaktionen die Ressourcen zentriert und von mehreren Diensten zugegriffen wird.

Listing 2.5. Attribute einer Aktivität in BPEL

```
<invoke name="berechnungMWST"
  partner="meinSteuerBerater" portType="ns:Buchhaltung"
  operation="mwst"
  inputVariable="preisEK" outputVariable="preisVK"/>

</invoke>
```

– **receive, reply und wait** Bei der Verbindung zwischen Prozessen, kommunizieren diese über eine Folge von *receive* und *reply* Aktivitäten. Während der Ausführung eines Prozesses blockiert bei receive der gesamte Ablauf bis die für ihn bestimmte SOAP-Nachricht eintrifft. Bei der *reply*-Aktivität wird daraufhin ein Ergebnis an den aufrufenden Partner zurückgegeben. Mithilfe der Aktivität *wait* wird ein Prozesse für ein bestimmtes Zeitintervall oder bis zu einem vorgegebenen Zeitpunkt angehalten. Dies geschieht mir der Angabe einer der beiden Attribute *for* oder *until*.

– **throw und terminate** Bei einem Workflow kann es vorkommen, dass bestimmte Bedingungen erfüllt sein müssen, um den korrekten Verlauf eines Prozesses gewährleisten zu können. Wie in Java mit dem Befehl throw, kann auch in BPEL das Auslösen von Fehlern mithilfe der Aktivität *throw* signalisiert werden (Siehe Listing 2.6). Die *terminate* Aktivität darf andere strukturierte Aktivitäten wie *FaultHandler* enthalten. Sie dient zur Abbrechung von Bearbeitungsschritten, wobei die Abarbeitung des Prozesses im nächsten übergeordneten Element fortgesetzt wird, in welches das *terminate* Element enthalten war.

– **scope und compensate** Nach dem Auftreten von Fehlern ist eine Fehlerbehandlung in BPEL durch die beiden Tags *scope* und *compensate* möglich. Nach der erfolgreichen Abarbeitung von Aktivitäten innerhalb eines durch *scope* definierten Gültigkeitsbereiches, können diese beim Auftreten von Fehlern mit dem *compensate* Element rückgängig gemacht werden. Wobei die *compensate*-Aktivität nur in einer Fehlerbehandlung oder einem „compensate handler“ (*compensate*-Anweisung) stehen darf.

Listing 2.6. Werfen einer Ausnahme

```
<throw faultName="ErrorConnectionServer" />

<wait for=''PT30M'' />

<wait until=''2008-06-08T12:15+03:00'' />
```

Die strukturierten Aktivitäten setzen sich aus anderen Aktivitäten zusammen und beschreiben einerseits den Datenfluss und andererseits die Geschäftslogik eines Prozesses. Für die sequenzielle Ablaufsteuerung wie Reihenfolgen, Schleifen-Konstrukten und mehrfachen Verzweigungen kommen die *sequence*-, *while*- und *switch*-Aktivitäten zum Einsatz. Die *flow*-Aktivität führt Aktivitäten parallel aus und wartet bis alle Kinder fertig sind. Die *pick*-Aktivität führt eine nicht-deterministische Auswahl zwischen mehreren Aktivitäten durch und legt fest, was zuerst aufgrund externer Ereignisse auszuführen ist. Für detaillierte Informationen zu den verschiedenen Aktivitäten sei auf [OASI07] verwiesen.

Kommunikation. Wie in vorherigen Paragraphen angedeutet ist, interagiert ein BPEL Prozess mit anderen externen Prozessen. Dazu deklariert die Sprache BPEL Partnerbeziehungen (Partner Links). Man unterscheidet zwei Arten der Interaktionen :

- Der BPEL Prozess ruft Operationen in anderen Diensten auf.
- Der BPEL Prozess empfängt Aufrufe von Klienten. Wobei einer der Klienten der Benutzer des Prozesses ist, der den ersten Aufruf durchgeführt hat. Die anderen Klienten sind Dienste, die aufgerufen sind, und Rückrufoperationen zurückgeben.

PartnerLinks können nicht nur Verknüpfungen zu den Diensten die im Geschäftsprozess involviert sind, sondern auch zu den Klienten die diesen Aufrufen sein. Ein BPEL Prozess hat mindestens eine Verknüpfung zu einem Klienten, weil es ein Klient geben muss der den Geschäftsprozess aufruft.
Normalerweise enthält ein BPEL Prozess mindestens einen aufgerufenen Partner Verknüpfung, weil er wahrscheinlich mindestens einen Dienst aufrufen wird. BPEL behandelt Klienten wie *partnerLink*s aus zwei Gründen. Der erste und offensichtliche Grund ist es, um asynchrone Operationen zu unterstützen. Der zweite Grund liegt darin, dass BPEL sich auf das Anbieten von Diensten konzentriert. Diese sind durch *portType*s, diese sind aus WSDL bekannt [W3C12] und gruppieren Nachrichten zu logischen Operationen, angeboten und können daher von mehr als einem Prozess benutzt werden. Dadurch kann in einem Geschäftsprozess zwischen verschiedenen Klienten unterschieden werden und jeder die Funktionalität, die ihm erlaubt ist, anbieten.

3 Workflow Management

3.1 Was sind Workflows

Im Umfeld von Unternehmen erfordern Geschäftsprozesse vorhandene Dienste zu integrieren, um neue, komplexere Dienste aufzubauen. Diese zusammengesetzten Dienste weisen neben dem geschäftlichen Mehrwert, im Vergleich zum nicht komponierten Dienst auch einen technischen Mehrwert auf. Ein komponierter Dienst hat die Eigenschaft, flexibler gegenüber Änderungen bezüglich seines Ablaufs zu sein. Das heißt bei Änderungen im Prozess wird er einfach neu komponiert. Typische Beispiele für Geschäftsprozesse sind Einkauf, Berichterstattung, Rekrutierung oder auch Call-Center-Prozesse.
Die *Workflow-Management Coalition* (WfMC) ist ein Verbund aus internationalen Organisationen von Workflow Herstellern, Nutzern, Beratern und Wissenschaftlern. Sie definiert einen *Workflow* als :"The computerised facilitation or automation of a business process, in whole or part."[Stan95]. Ein Workflow beschreibt dann die Reihenfolge einer Reihe von Aktivitäten, die von verschiedenen Agenten ausgeführt werden, um eine gegebene Prozedur innerhalb einer Organisation durchzuführen. Das Ziel eines Workflows ist es, die Automatisierung des gesamten Prozesses anstatt isolierter Aktivitäten. Ursprünglich beschäftigt sich ein Workflow mit der Übergabe einer Tätigkeit von einem Mitarbeiter zum anderen, zusichernd dass der letzte seine Aufgabe starten kann. Heute sind, mit dem stetig ansteigenden Vernetzungsgrad von Systemen Workflows mehr und mehr im Mittelpunkt, deren Akteure nicht mehr nur Menschen, sondern auch Maschinen sind. Je nach Umfang und Häufigkeit der Anwendung von Workflows lassen sich drei Arten unterscheiden:

- **Ad-hoc Workflow** In dieser Art von Workflows wird der Prozess zur Laufzeit vom Anwender definiert, verwaltet und abgearbeitet. Trotz des hohen Freiheitsgrades enthalten sie sehr viele Ausnahmen.

- **Strukturierter Workflow** Hier werden Prozesse in einem Prozess-Besitzer definiert und verwaltet. Wobei der Benutzer nicht am Prozess teilnimmt.

- **Flexibler Workflow** Zusätzlich zum strukturierten Workflow nimmt der Anwender am Prozess teil und steuert/beeinflusst diesen.

3.2 Workflow-Management-Systeme

Die Unternehmen konzentrieren sich in großer Ebene auf ihrer Kernkompetenzen. Heute mit dem großen Einsatz der Informationstechnik können Prozesse besser optimiert werden. Diese Prozesse können dann durch sogenannte Workflow-Managementsysteme (WfMS) umgesetzt werden. Workflow-Managementsysteme stellen dann die Ausführungsumgebungen bereit und koordinieren die Interaktion der beteiligten Dienste sowie auch die Speicherung der für die Abarbeitung notwendigen Daten. Um eine weitestgehende Systemunabhängigkeit und Interoperabilität zu erreichen, setzt sich die WfMC das Zeil ein Architekturmodell

für solche Workflow-Managementsysteme zu entwerfen.
Das in Abbildung 6 angezeigte Modell stellt eine wichtige Standardisierung für

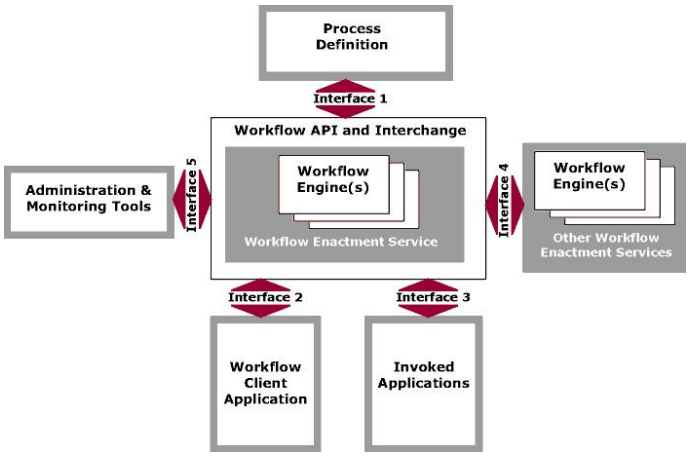


Abbildung 6. WfMC-Workflow-Referenz-Modell (Quelle : [WfMC])

viele Workflow Produkte und WfMS dar. Im Folgenden werden die Komponenten und Schnittstellen kurz vorgestellt:

- **Process Definition** Diese Komponente ist für die Modellierung von Prozessen zuständig. Die darin stehende Schnittstelle (Workflow Definition Interchange) bietet ein Austauschformat für viele Daten. Hierzu gehören:
  - Identifikation von Aktivitäten innerhalb eines Prozesses.
  - Informationen über Ressourcenallokationen.
  - Identifikation von Datentypen.

- **Workflow Client Application** ist ein Programm, das eine Schnittstelle mit der zur Modellierung notwendigen Funktionalität zur Verfügung stellt. Mittels dieser Anwendung interagiert der Workflow-Modellierer mit dem Workflow-Managementsystem.

- **Invoked Applications** Mithilfe dieser Komponente werden externe Anwendungen und somit auch ihre Funktionen zu einer Aktivität aufgerufen und eingebunden.

- **Other Workflow Engines** Ein WFMS sollte in der Lage sein mit anderen Workflow-Managementsystemen zu kommunizieren und die angebotenen Prozesse zu nutzen. In diesem Kontext erlaubt die Schnittstelle zwischen zwei WFMS die Interoperabilität mehrerer Komponenten in unterschiedlichen Systemen. Einerseits ist es möglich durch den Austausch von Prozessdefinitionen und konkreten Typen, dass ein Prozess in erster Ebene in

Laufzeitumgebung A startet und in Laufzeitumgebung B weiterverarbeitet wird. Andererseits lässt sich mit wenig Aufwand und Synchronisation ein Workflow aus der Prozessdefinition der ersten Umgebung in einer zweiten Umgebung ausführen.

– **Administration & Monitoring Tools** Mit den Admistrations- und Überwachungswerkzeuge lassen sich die Laufzeitumgebungen verschiedener Hersteller systemunabhängig überwachen. Dadurch werden Fehlersituationen während der Laufzeit von Prozessen protokolliert.
–

3.3 XML Process Definition Language (XPDL)

Definition. Die XML Process Definition Language kurz XPDL ist eine XML-basierte Sprache, die durch Workflow-Mangementsysteme ausgeführt wird. Sie ist der größte Konkurrent zu BPEL. Zuständig für ihre Spezifikation[Stan02] ist die WfMC. Das Ziel von XPDL ist es eine Verkehrsprache für den Bereich von Workflow zu entwickeln. Es sollen Werkzeuge übergreifend Funktionalitäten für Importieren und Exportieren von Prozessdefinitionen und zur Modellierung und Simulation der Prozesse unterstützt werden (siehe Abbildung 7).

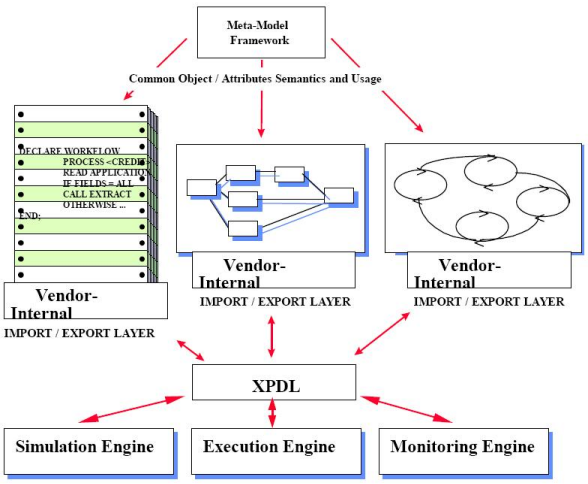


Abbildung 7. Das Konzept des Process Definition Interface (Quelle : [Stan02])

XPDL Metamodell. Das in Abbildung 8 dargestelle XPDL-Metamo- dell beschreibt den Zusammenhang zwischen den einzelnen Sprachelementen. Es enthält

statische Entitäten wie Daten und Anwendungen und dynamische Konzepte wie z.B. Prozesse.
Für die Repräsentation der statischen Entitäten werden folgende Metadaten verwendet:

- **Workflow-relevante Daten** (*Workflow Relevant Data*): Nachdem diese Daten initialisiert, erzeugt und von anderen Anwendungen gelesen wurden, können sie dann im Laufe des Workflows verwendet werden.

- **Workflow-Teilnehmer-Spezifikation** (*Workflow Participant Specification*): Sie dient zur Beschreibung der in einem Prozesschritt auszuführenden Ressourcen. Diese Spezifikation repräsentiert eine von einer oder mehreren Person auszufüllenden abstrakten Rolle oder Ressource.

- **Workflow-Anwendungsdeklaration** (*Workflow Application Declaration*): Da die Workflow-Prozessen andere Anwendungen zu ihrer Ausführung benötigen, lassen sich durch diese Komponente diese Softwarekomponenten beschreiben. Die Initialisierung dieser Anwendungen übernimmt die Workflow-Engine und die Workflow-relevante Daten werden dabei als Parameter übergeben.

Die dynamischen Aspekte im XPDL-Metamodell werden durch folgende Entitätstypen beschrieben:

- **Transitionsinformation** (*Transition Information*): Dadurch wird der Kontrollfluss zwischen den einzelnen Aktivitäten beschrieben. Die Basisstruktur einer Transitionsinformation besteht aus einer Anfang- und Endaktivität sowie einer Bedingung, die für die Ausführung der Transaktion den Ausschlag gibt. In XPDL gibt es neben der Steuerung der Verzweigungen mit XOR, OR oder UND, vier andere Bedingungen. Diese sind *CONDITON*, *OTHERWISE*, *EXCEPTION* und *DEFAULTEXCEPTION* (für Details siehe [Stan05]).

- **Workflow-Prozessaktivität** (*Workflow Process Activity*): Darunter versteht man einen Arbeitschritt, der von den Teilnehmern (*Participant*) ausgeführt wird. Jede Arbeitseinheit ist in den beiden Fällen, die automatische Ausführung durch das WfMS als auch die Ausführung durch einen Workflow-Teilnehmer, durch einen Anfangs- und Endpunkt bestimmt. Es gibt drei konkrete Subtypen einer Workflow-Prozessaktivität, diese sind die Blockaktivität (*Block Activity*), die einzelne Aktivität (*Atomic Activity*) und die Subprozessdefinition (*Sub-Process Definition*).

4 Vergleich zwischen XPDL und BPEL

Im Bereich der Modellierungssprachen von Geschäftsprozessen sind in den letzten Jahren viele Studien und Vergleiche gemacht worden. Es werden reine Vergleiche erstellt [Shap02], Verbesserungen vorgeschlagen [vdAa05] [HHAM05] und Spezifikationen evaluiert [GoJW04].

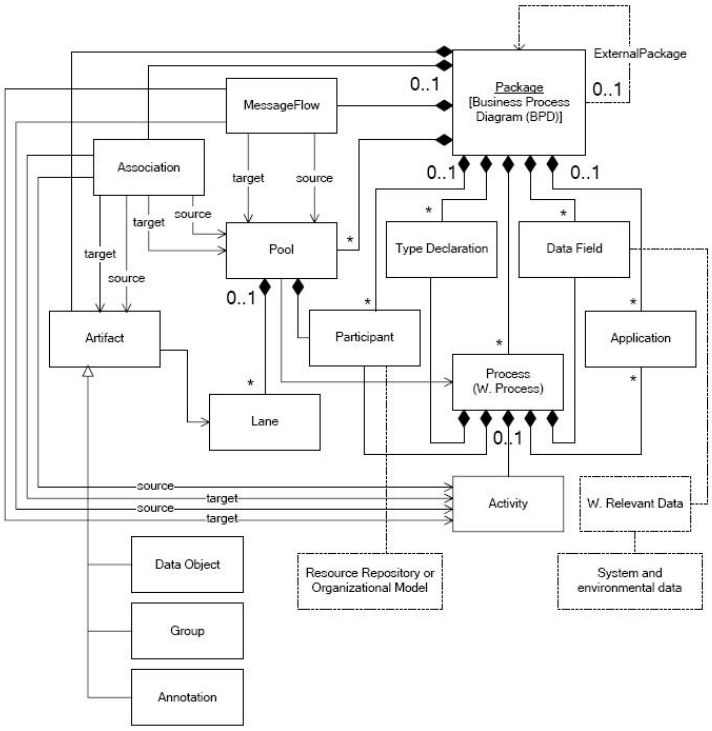


Abbildung 8. XPDL Metamodell (Quelle : [Stan05])

4.1 Technische Sicht

Auf der technischen Ebene haben beide Sprachen XPDL und BPEL deutliche Unterschiede. Diese liegen in der Art der Beschreibung von Geschäftsprozessen und wie genau diese Prozesse in der jeweiligen Sprache dargestellt werden.
BPEL ist eine Block-basierte Sprache und XPDL als eine Graph-basierte Sprache entworfen wurde. Der Unterschied zwischen den beiden Ansätzen ist, dass einerseits die Blockstruktur ein besserer Lösungsansatz für Programmiersprachen ist, aber Schwierigkeiten in der Beschreibung von graphischen Abläufen bereitet. Andererseits werden Graphen und Diagramme für die Darstellung von Prozessabläufen von Anwendern bevorzugt. Außerdem lassen sich große Prozessabläufe in BPEL nur mit komplexen sprachlichen Formulierungen darstellen. Ein anderer Vorteil erbringt XPDL bei der Verwendung von BPMN (Die Business Process Modeling Notation). BPMN is definiert als: „eine grafische Spezifikationssprache in der Wirtschaftsinformatik. Sie stellt Symbole zur Verfügung, mit denen Fach- und Informatikspezialisten Geschäftsprozesse und Arbeitsabläufe (techn.: Workflows) modellieren können“[W3C]. Für detaillierte Informationen zu den syntaktischen Unterschieden zwischen den beiden Sprachen sei auf [Shap02] verwiesen.

Eine andere Form für den technischen Vergleich von XPDL und BPEL ist ihre Implementierung. Eine Vielzahl von Software Herstellern hat ihre Produkte zu einem der beiden Geschäftsprozess-Standards kompatibel gemacht. Hier ist eine Menge dieser Hersteller aufgelistet:

- **BPEL**
  - Active Endpoints
  - Apache Ode
  - IBM WebSphere Process Server
  - JBoss JBPM
  - Microsoft Biztalk Server
  - Microsoft Windows Workflow Foundation
  - Oracle BPEL Process Manager
  - SAP NetWeaver Exchange Infrastructure
  - Sun SeeBeyond eInsight Business Process Manager
  - SEEBURGER Business Integration Server
- **XPDL**
  - Bonita
  - Cape Visions
  - IBM Filenet Business Process Manager 4.0
  - Oracle 9i Warehouse Builder
  - Software AG's Crossvision BPM
  - Together Workflow Editor
  - WfmOpen
  - Intalio

Die Liste der BPEL Implementierungen enthält mehr große Software Herstellern als die Liste der XPDL Implementierungen. Dies kann daran liegen, dass diese BPEL als prominenteren Standard annehmen, und dadurch einen größeren Markt ansprechen wollen. Die Anwendungen, die XPDL unterstützen, lassen sich grundsätzlich in drei Arten einteilen. Es gibt ein Typ von Anwendungen, die allein die Dokumentation von Geschäftsprozessen unterstützt, der andere führt im Sinne des WfMC-Referenzmodells die Prozesse als Workflow Enactment Engine aus und der dritte kann diese Prozesse simulieren und analysieren.

4.2 Betriebliche Sicht

Die Grundidee hinter neuen Technologien und Plattformen für die Entwicklung von Unternehmensanwendungen ist es, eine Entwicklungsumgebung für Geschäftsprozesse zu entwerfen, die einfach zu bedienen ist. Solche Geschäftsanwendungen können heutzutage jedoch nicht mehr isoliert behandelt werden, und müssen mit anderen Unternehmen interagieren. Für den Anwender innerhalb des Unternehmens ist es von großer Bedeutung die Geschäftsprozesse mit einfachen Modellierungsanwendungen zu entwerfen und darzustellen. In diesem Sinne gibt es für die Sprache XPDL eine Reihe von Modellierungsanwendungen, zu zählen sind z.B. COSA BPM, Fujitsu Interstage Business Process Manager und andere. Außerdem lassen sich XPDL-Packages in Simulatoren einlesen und

in ihrem Verhalten simulieren. Diese Simulatoren können für die Unternehmen nicht nur Kosten beim Aufbau von großen Projekten reduzieren sondern auch dem Entwickler dabei helfen, in einer echten Entwicklungsumgebung zu arbeiten. Heute wird stark in diesem Gebiet erforscht und entwickelt um Funktionen und WfMS-Lösungen zur Unterstützung bei der Ausführung von Geschäftsprozessen anzubieten. Allerdings werden heute BPEL Systeme als zentrale Funktion im Business Prozess Management bezeichnet. Sie decken technische Aspekte, die nicht von den WfMS unterstützt werden ab. Außerdem haben die BPEL Prozesse eine große Verfügbarkeit, da sie auf Servern laufen.
Die Realisierung einer robusten Dienstbereitstellung für die Benutzer und das Geschäft von großer Bedeutung. Für die viele Unternehmen stellt sich dieser als eine große Herausforderung dar. Häufig beteiligen sich im Lieferungsverlauf von Diensten Aktivitäten von mehreren Agenten und stellen dazu einen Koordinationsaufwand auf. Der Bedarf nach der Koordination von Geschäftsprozessen in verhältnismäßig autonomen Agenturen ist in den letzten Jahren gestiegen.

4.3 Zusammenfassung

Trotz den Unterschieden im Einsatz der Modellierung von Geschäftsprozessen, stellt sich die Verbindung der beiden Technologien als eine Herausforderung dar. Doch sollten Workflow-Managementsysteme eine Integration von Geschäftsprozessen erlauben, die auf BPEL basieren. Einerseits könnten WfMS BPEL ausnutzen, um notwendige Informationen für die Bearbeitung von Geschäftsprozessen bereitzustellen. Andererseits nutzen BPEL Prozesse die WfMS um die Bearbeitung eines nicht technischen Arbeitsschritt zu ermöglichen.

5 Fazit

Die Service Orientierte Architektur bietet in ihrer Umsetzung in der Informationstechnologie starke Vorteile. Zu zählen ist das Senken der IT-Kosten und die Möglichkeit der Wiederverwendung. Außerdem wird einerseits durch die Verwendungen von Services und ihrer Neukombination eine hohe Flexibilität erreicht. Andererseits reduziert sich die Komplexität durch Kapselung der Implementierungsdetails hinter den Services.
Diese Vorteile kommen aber nur zur Geltung, wenn Geschäftsmodell und Geschäftsprozesse serviceorientiert sind. Andere Schwierigkeiten in der Umsetzung von SOA können auch der Mangel an IT-Spezialisten mit fundiertem betriebswirtschaftlichen Know How, architektonische Herausforderungen oder falsche Vorstellungen durch SOA-Hype. Trotz allen Schwierigkeiten bleibt spannend: was die Zukunft im Einsatz von SOA bringt und wie die beiden Sprachen XPDL und BPEL mit anderen Standard und Herstellern konkurrieren.

Literatur

ADHR+05. W. M. P. Aalst, M. Dumas, A. H. M. Hofstede, N. Russell, H. M. W. Verbeek und P. Wohed. Life After BPEL? S. 35–50. 2005.

Erl08. Thomas Erl. *SOA Principles of Service Design.* Prentice Hall PTR Upper Saddle River, NJ, USA. 2008.

GAPJV+08. Florian Gottschalk, van der Aalst, Wil M. P., Monque H. Jansen-Vullers und Marcello La Rosa. Configurable Workflow Models. *International Journal of Cooperative Information Systems*, Januar 01 2008.

GoJW04. Jeffrey Gortmaker, Marijn Janssen und René W. Wagenaar. The advantages of web service orchestration in perspective. In *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*, New York, NY, USA, 2004. ACM, S. 506–515.

HHAM05. ter Hofstede, Arthur H., van der Aalst und Wil M. YAWL: yet another workflow language, Juni 01 2005.

Josu07. Nicloai M. Josuttis. *SOA in Practice.* O Reilly Media, Inc. 1005 Gravenstein Highway North, Sebastopol, CA 95472. 2007.

Newc05. Eric Newcomer. *Understanding Web Services : XML, WSDL, SOAP and UDDI.* Addison-Wesley. 2005.

Note04. W3C Working Group Note. Web Services Architecture, 11 February 2004.

OASI07. OASIS. Web Services Business Process Execution Language Version 2.0, 11 April 2007.

Pelt03. Chris Peltz. Cover Feature: Web Services Orchestration and Choreography. *j-COMPUTER*, 36(10), oct 2003, S. 46–52.

Shap02. R. Shapiro. A Comparison of XPDL, BPML and BPEL4WS (Version 1.4). 2002.

srcs06. soa-rm cs. Reference Model for Service Oriented Architecture 1.0, 2 August 2006.

Stan02. Workflow Management Coalition Workflow Standard. Workflow Process Definition Interface – XML Process Definition Language, October 25, 2002.

Stan05. Workflow Management Coalition Workflow Standard. Workflow Process Definition Interface – XML Process Definition Language v2.0, 3 October 2005.

Stan95. Workflow Management Coalition Workflow Standard. Workflow Management Coalition- The Workflow Reference Model, 19-Jan-95.

vdAa05. Wil M.P. van der Aalst. Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language, 2005.

W3C. W3C. W3C.

W3C12. W3C. Web Services Description Language (WSDL) 1.1, 15 March 20012.

WDB.05. I. Melzer W. Dostal, M. Jeckle und B.Zengler. *Service-orientierte Architekturen mit Web Services : Konzepte - Standards - Praxis.* ELSEVIER : Spektrum Akademiker Verlag. 2005.

WfMC. WfMC. WfMC.

WRNi03. M. Hadley N. Mendelsohn J-J. Moreau W3C Recommendation, M. Gudgin und H. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework, 24 June 2003.

Nichtfunktionale Anforderungen im Requirements Engineering

Jörn Kuhlenkamp

Betreuer: Christoph Rathfelder

Zusammenfassung Obwohl die Betrachtung von NFA von wichtiger Bedeutung für den Erfolg eines Softwareprodukts ist, finden sie in vielen Entwicklungsmethoden häufig zu geringe Beachtung.
Es existieren bis dato nur wenige Arbeitsansätze, die eine umfassende Definition, Analyse und Dokumentation von NFA ermöglichen.
Für potentielle Anwender ist allerdings bereits unter diesen die Auswahl des geeigneten Ansatzes aufgrund fehlender Bewertungskriterien problematisch. Zudem wurden sie noch nicht in einen übersichtlichen Bezug zueinander gesetzt.
Im Rahmen dieser Arbeit werden Kriterien für die Bewertung von Arbeitsansätzen für NFA herausgearbeitet und auf dieser Basis verschiedene Methoden für den Umgang mit NFA aus bestehenden Arbeiten zusammengetragen, bewertet und schließlich auf ihren Nutzen hin miteinander verglichen. Ein besonderer Fokus wurde dabei auf die Verwendung des *NFA-Frameworks* gelegt, dass einen der am umfassendsten entwickelten Ansätze darstellt.

1 Einleitung

Diese Arbeit gibt eine Einführung in das Gebiet der NFA innerhalb des Requirements Engineering. NFA tragen maßgeblich zur Qualität eines (fertigen) Softwareprodukts bei. Ihr Nichtbeachten führt häufig zu Fehlern, die nur aufwendig zu beheben sind und hohe Kosten verursachen [MyCN92, S.1]. Trotz dieser Tatsache, betrachten viele Ansätze, für die Arbeit mit NFA, diese nicht über den kompletten Entwicklungszeitraum hinweg oder analysieren und dokumentieren sie nicht ausreichend. Das *NFA-Framework* [MyCN92] ermöglicht es NFA detailliert zu betrachten, indem bereits erhobene Anforderungen verfeinert und in verschiedene untergeordnete Anforderungen aufgeteilt werden. Dann werden die Auswirkungen von alternativen Entwurfsentscheidungen auf einzelne NFA analysiert und schließlich eine Menge von Entwurfsentscheidungen auf der Basis dieser NFA ausgewählt und dokumentiert.
Die Nutzung des NFA-Frameworks in der *zielorientierte Anforderungsanalyse* [MyCY99,MCLW$^+$01] ermöglicht es, diese Entwurfsentscheidungen gleichermaßen auf der Basis von FA und NFA Anforderungen zu treffen, da beide für den Erfolg eines Softwareprodukts von Bedeutung sind.
Ein alternativer Ansatz, der zusätzlich bereits eine vollständigere Erhebung von

FA und NFA in Verbindung mit dem NFA-Framework ermöglicht, ist der *LEL-Ansatz* [CyLe01]. In diesem werden FA und NFA jedoch getrennt analysiert und Ergebnisse an bestimmten Stellen des Entwicklungsprozesses miteinander kombiniert.
Die mögliche Nutzung des NFA-Frameworks in verschiedenen existierenden und zukünftigen Methoden und Ansätzen macht es notwendig, diese kritisch zu bewerten, um in Abhängigkeit von einer gegebenen Problemstellung eine geeignete (Kombination von) Ansätzen(en) auszuwählen.
Der Aufbau der vorliegenden Arbeit ist wie folgt strukturiert: Kapitel 2 beschreibt die Grundlagen des Themengebiets und die Motivation NFA innerhalb des Requirements Engineering auf eine besondere Art und Weise zu betrachten. Zu diesem Zweck werden in Kapitel 2.1 FA und NFA voneinander abgegrenzt und in Kapitel 2.2 die Bedeutung von NFA beschrieben. Kapitel 2.3 gibt neben der verbreiteten technischen eine alternative risikoorientierte Definition des Begriffs Requirements Engineering, die die Notwendigkeit einer frühzeitigen Betrachtung von NFA besser veranschaulicht. Daraufhin klassifiziert Kapitel 2.4 Ansätze für die Arbeit mit NFA horizontal in *produkt-* und *prozessorientierte Ansätze*, die NFA entweder nachträglich validieren oder über den kompletten Entwicklungsprozess hinweg betrachten und vertikal in quantitative und qualitative Ansätze. Kapitel 2.5 beschreibt Kriterien einer guten Anforderungsspezifikation. Dies verdeutlicht im Verlauf der Arbeit, dass im Bezug auf NFA viele dieser Kriterien durch den Einsatz geeigneter Analyse- und Dokumentationsmethoden wie dem NFA-Framework besser erfüllt oder erst erfüllbar werden. Kapitel 2.6 geht schließlich näher auf Problemfaktoren bei der Arbeit mit NFA ein.
Kapitel 3 stellt das NFA-Framework vor. Dafür wird in Kapitel 3.1 eine Einführung und in Kapitel 3.2 eine Definition der Grundlagen des Frameworks gegeben. Daraufhin wird in Kapitel 3.3 auf seine praktische Anwendung und in Kapitel 3.4 auf seine Bewertung eingegangen.
Kapitel 4 beschreibt die Nutzung des NFA-Frameworks in der zielorientierte Anforderungsanalyse. Dabei geht Kapitel 4.1 auf die Motivation der Nutzung und Kapitel 4.2 wiederum auf die benötigten Grundlagen, Kapitel 4.3 auf die Anwendung und Kapitel 4.4 auf die Bewertung der Methode ein.
Der alternative LEL-Ansatz wird in Kapitel 5 vorgestellt. Kapitel 5.1 beschreibt das LEL, dass die Erhebung von FA und NFA erleichtert, dokumentiert und den Ausgangspunkt für deren weitere Analyse darstellt. Die Anpassung des NFA-Frameworks für den LEL-Ansatz wird in Kapitel 5.2 und die Anwendung in Kapitel 5.3 beschrieben. Nach der Bewertung des Ansatzes (Kapitel 5.4) erfolgt das abschließende Fazit der Arbeit und es wird ein Ausblick auf zukünftige Entwicklungen innerhalb des Themengebiets gegeben (Kapitel 6).

2 Grundlagen

Bei der systematischen Entwicklung von Software spielt das Requirements Engineering eine wichtige Rolle. Die Qualität des Endprodukts wird dabei sowohl von funktionalen- als auch von nichtfunktionalen Anforderungen bestimmt [SuCh05,

S.1]. Da Abhängigkeiten zwischen beiden Anforderungsarten bestehen, ist es allerdings auch notwendig, sie gemeinsam zu betrachten. Um unterschiedliche Ansätze für NFA zu vergleichen, müssen zuerst die allgemeinen Ziele des Requirements Engineerings und die besonderen Eigenschaften von NFA verstanden werden. Zudem ist es notwendig geeignete Bewertungskriterien und vorhandene Problemstellungen zu identifizieren.

2.1 Funktionale- und nichtfunktionale Anforderungen

Bei der Entwicklung eines Softwaresystems spielt die Funktionalität eine bedeutende Rolle. Durch funktionale Anforderungen wird bestimmt und festgehalten, welche konkreten Aufgaben ein System erfüllen soll. Stellen wir uns für eine Beispielsanwendung einen Veranstaltungskalender vor. Unter die FA fallen Aufgaben wie das Hinzufügen, Löschen und Bearbeiten einzelner Veranstaltungen.

Definition: „Eine *funktionale Anforderung* definiert eine vom System bzw. von einer Systemkomponente bereitzustellende Funktion oder einen bereitzustellenden Service."[Pohl08, S.15]

Dabei ist der Systembegriff wie folgt definiert:

Definition: Ein *System* ist „eine zusammengehörige, von ihrer Umgebung abgrenzbare Menge von Komponenten, die durch koordiniertes Zusammenwirken Leistungen erbringen" [Glin06, Foliennummer 3].

Oft werden allerdings zusätzlich bestimmte Qualitätseigenschaften von Softwaresystemen erwartet oder allgemeine Restriktionen, die es zu erfüllen gilt, gestellt. Dabei kann es sich beispielsweise um Sicherheits-, Leistungs- und Qualitätsmerkmale handeln. Diese Eigenschaften werden als *nichtfunktionale Anforderungen (NFA)* bezeichnet. Für das zuvor genannte Beispiel kann dies eine Anforderung an die Reaktionszeit sein, die fordert, dass eine gestellte Anfrage in weniger als zwei Sekunden beantwortet wird.

Definition: Eine *NFA* definiert eine qualitative Eigenschaft des gesamten Systems, einer Systemkomponente oder einer Funktion. (In Anlehnung an [Pohl08, S.16])

Robertson stellt funktionale und nichtfunktionale Eigenschaften mit folgendem Beispiel in Beziehung: „Diese Eigenschaften werden nicht benötigt, weil sie grundlegende Aktivitäten des Produkts sind [...], sondern sie sind da, weil der Nutzer möchte, dass die grundlegenden Aktivitäten auf eine bestimmte Art und Weise ausgeführt werden." [RoRo99, S.112]

2.2 Bedeutung von nichtfunktionalen Anforderungen

Die Bedeutung von NFA in der Softwareentwicklung nimmt stetig zu. Dies liegt zum Einen daran, dass nach der Fertigstellung eines Softwaresystems, Fehler aufgrund mangelhafter Erhebung und Beachtung von NFA zu den teuersten und am schwersten zu behebenden gehören [MyCN92, S.1]. Des Weiteren werden Eigenschaften wie Verfügbarkeit, Sicherheit und Benutzerfreundlichkeit immer stärker fokussiert. Wie im Bereich des Kundepflegemanagements (eng. Customer Relationship Management, *CRM*) oder des One-to-One Marketings [PeRD00], welches auf einen hohen Individualisierungsgrad von Kundenbeziehungen abzielt. Dieser ist nur durch die Nutzung von IT-Systemen zu realisieren, die hohe Anforderungen an Zugänglichkeit und Nutzbarkeit erfüllen. Trotz solcher Entwicklungen fanden NFA in der Literatur lange vergleichsweise wenig Aufmerksamkeit [CydPL04, S.1], bis ihr Einfluss auf den Entwicklungsprozess vermehrt wahrgenommen wurde [ChNi95, S.1]. Allerdings sind viele Fragestellungen für den Systematischen Umgang mit NFA noch ungeklärt [CNYM99].

2.3 Diskussion des Begriffs Requirements Engineering

Der Begriff der *Requirements Engineering* ist weit gefasst. Um ihn zu verdeutlichen sollen an dieser Stelle zwei unterschiedliche Definitionen gegeben werden.

Definition: „Das *Requirements Engineering* ist ein kooperativer, iterativer, inkrementeller Prozess, dessen Ziel es ist zu gewährleisten, dass (1) alle relevanten Anforderungen bekannt und in dem erforderlichen Detaillierungsgrad verstanden sind, (2) die involvierten Stakeholder (Anspruchsberechtigten) eine ausreichende Übereinstimmung über die bekannten Anforderungen erzielen und (3) alle Anforderungen [...] dokumentiert bzw. konform zu den *Spezifikationsvorschriften* spezifiziert sind." [Pohl08, S.43]

Aufgrund dieser Definition lassen sich verschiedene Kriterien identifizieren, die auch eine Methode für die Arbeit mit NFA - da sie ein Teil der allgemeinen Anforderungen sind - erfüllen sollte. Die Menge der Anforderungen sollte daher vollständig erhoben und ausreichend detailiert analysiert werden. Des Weiteren sollten die verschiedenen Anspruchsberechtigten eine ausreichende Übereinstimmung über die bekannten Anforderungen erzielen. Dafür ist es jedoch in der Praxis notwendig, zuvor festzustellen, in welcher Beziehung verschiedene Anforderungen zueinander stehen und das Kosten/Nutzenverhältnis zwischen verschiedenen Alternativen abzuwägen (Trade-Off-Entscheidungen). Der letzte Punkt verdeutlicht, dass die Anforderungen eines Softwaresystems der Spezifikationsvorschrift genügen sollten. Dabei ist eine *Spezifikation von Anforderungen* wie folgt definiert:

Definition: Eine Spezifikation von Anforderungen für Software (eng. Software Requirements Specification, *SRS*) ist ein dokumentiertes Produkt der Anforderungserhebung, -analyse und -validierung (siehe [oEEn98, S.1ff]), sowohl für

FA und als auch für NFA.

Eine weitere, besonders im Umgang mit *NFA* an Bedeutung gewinnende, Definition ist die risikoorientierte.

Risikoorientierte Definition: Die Anforderungstechnik ist die „Spezifikation und Verwaltung von Anforderungen mit dem Ziel, das Risiko zu minimieren, dass ein *System* entwickelt wird, welches den Kunden nicht nützt oder gefällt" [Glin06, Foliennummer 5].

Diese Definition verdeutlicht dabei die Notwendigkeit NFA über den gesamten Entwicklungsprozess einzubeziehen, um frühzeitig Konflikte zu finden und zu lösen. Da beide Definitionen nicht im Konflikt zueinander stehen, soll innerhalb dieser Arbeit, die risikoorientierte Definition des Begriffs Requirements Engineerings als Erweiterung der Ursprünglichen angesehen werden. Diese viele Aspekte einbeziehende Betrachtungsweise begründet unter Anderem die Entstehung von verschiedenen Ansätze für die Arbeit mit NFA.

2.4 Produkt- und prozessorientierter Ansatz

Ansätze für die Arbeit mit NFA werden entweder als produktorientiert oder prozessorientiert bezeichnet [MyCN92, S.1].
Der *produktorientierte Ansatz* versucht NFA in messbaren Einheiten zu fassen, um nach der Fertigstellung eines Systems zu überprüfen, ob es sich in den zuvor festgesetzten Spezifikationen befindet [Fran98, S.1]. So kann beispielsweise durch die Aussage „Die Funktion XY liefert innerhalb von 0,5 Sekunden eine Antwort" eine NFA an die Performanz eines Sofware-Systems spezifiziert werden.
Beim *prozessorientierten Ansatz* werden NFA bereits während des Software-Entwicklungsprozesses einbezogen. Dabei wird davon ausgegangen, dass sich Entscheidungen bezüglich des Software-Entwurfs positiv oder negativ auf NFA auswirken können [MyCN92, S.1]. Die Literatur konzentrierte sich eine lange Zeit bevorzugt auf den produktorientierten Ansatz [CydPL04, S.1]. Erst später rückte der prozessorientierte Ansatz aus den zuvor genannten Zielen (siehe Kapitel 2.3) in das Zentrum des Interesses.
Die Informationen im folgenden Absatz stammen aus [MyCN92, S.1ff.]. Des Weiteren ist es möglich, den Umgang mit NFA neben der horizontalen Einteilung in produkt- und prozessorientierte Ansätze, vertikal in quantitative und qualitative Ansätze zu unterteilen. Dabei versuchen *quantitative Ansätze* NFA in messbaren Einheiten zu fassen und finden daher bevorzugt mit produktorientierten Ansätzen Verwendung. Der prozessorientierte Ansatz legt einen Fokus auf den qualitativen Umgang mit NFA, ist allerdings nicht auf diesen beschränkt. Besonders unter Einbeziehung der risikoorientierten Definition des Requirements Engineering ist fraglich, ob ein produktorientierte Ansatz alleine die beste Möglichkeit zur frühzeitigen Risikobewertung darstellt.
Allerdings sollten quantitative produktorientierte und qualitative prozessorien-

tierte Ansätze nicht als gegensätzlich, sondern als sich ergänzend betrachtet werden.

2.5 Kriterien einer guten Anforderungsspezifikation

Wie in Kapitel 2.3 verdeutlicht, hängt die Qualität eines Vorgehens innerhalb des Requirements Engineerings auch von der Qualität der Anforderungsspezifikation ab. Diese sollte sich wiederum nach bestimmten Kriterien richten, die in diesem Kapitel verdeutlicht werden sollen. In Verbindung, mit den in Kapitel 2.3 identifizierten Zielen, bilden diese Kriterien Bezugspunkte, anhand derer der Einsatz des NFA-Frameworks oder anderer Methoden konkret bewertet werden kann. Die in diesem Abschnitt folgenden Definitionen basieren auf den weit verbreiteten Standart des IEEE[oEEn98, S.1ff]. Zu den Charakteristiken einer guten *SRS* zählen demnach unter Anderem:

Korrektheit
: „Eine *SRS* ist korrekt, wenn [...] jede in ihr festgelegte Anforderung eine ist, die die Software einhalten soll."

Eindeutigkeit
: „Eine *SRS* ist eindeutig, wenn [...] jede in ihr festgelegte Anforderung nur eine Interpretationsmöglichkeit zulässt."

Vollständigkeit
: Eine vollständige *SRS* beinhaltet alle wichtigen Anforderungen und definiert das Verhalten für alle möglichen Klassen von Eingabedaten. In ihr enthaltende Elemente wie Tabellen oder Diagramme sind vollständig gekennzeichnet und referenziert. Alle verwendeten Begriffe und Maßeinheiten werden definiert.

Interne Konsistenz
: „Eine *SRS* ist intern konsistent, wenn [...] keine Teilmenge von individuellen Anforderungen einen Konflikt beschreibt."

Verifizierbarkeit
: „Eine *SRS* ist verifizierbar, wenn [...] jede in ihr aufgeführte Anforderung verifizierbar ist. Eine Anforderung ist verifizierbar, wenn [...] ein endlicher kosteneffektiver Prozess existiert, mit dem eine Person oder Maschine überprüfen kann, ob das Softwareprodukt die Anforderungen erfüllt."

Rückverfolgbarkeit
: „Eine *SRS* ist rückverfolgbar, wenn die Herkunft jeder ihrer Anforderungen klar ist und wenn sie die Referenzierung aller Anforderungen in der zukünftigen Entwicklung oder erweiterten Dokumentationen ermöglicht."

2.6 Problemfaktoren bei der Arbeit mit NFA

NFA sind übergeordnete Qualitätseigenschaften für ein Softwareprodukt. Dies kann ihre Validierung erschweren, besonders, wenn sie nicht quantitativ messbar sind. Bereits ihre Erhebung kann ein Problem darstellen, falls sich Wünsche von verschiedenen Anspruchsberechtigten (eng. Steakholder) überschneiden. Beispielsweise kann neben der Anforderung nach leichter Zugänglichkeit eine hohe Sicherheit gewünscht werden. Da NFA potentiell im Konflikt zueinander stehen [ErYM06, S.2], ist besonders auf ihre interne Konsistenz zu achten.
Häufig werden NFA bei der Anforderungserhebung und Analyse nur informell oder in natürlicher Sprache erfasst. Außerdem sind sie zwar in vielen Software-Entwicklungsmethoden vertreten, werden jedoch oft als untergeordnete Anforderungen betrachtet [CydPL02, S.1]. Dies führt häufig dazu, dass sie in Dokumentation verborgen bleiben und daher nicht beachtet oder vergessen werden [CydPL02, S.1]. Problematisch wird daher auch die Integration von NFA in einen konkreten Entwurf [CydPL04, S.1]. Außerdem kann ihre Rückverfolgbarkeit erschwert werden. Daher die Möglichkeit festzustellen, welche Entwurfsentscheidungen durch bestimmte NFA begründet werden. Dies kann besonders in der Wartungs- und Pflegephase eines Softwareprodukts zu Problemen führen. Daher sollten NFA sowohl zu Beginn als auch während des kompletten Lebenszyklus eines Softwareprodukts beachtet werden [CydPL02, S.1]. Diese und weitere Problemfaktoren bei der Arbeit mit NFA führten zu der Entstehung des im Folgenden vorgestellten NFA-Framework.

3 *NFA-Framework*

3.1 Einleitung

Das 1992 von John Mylopoulos, Lawrence Chung und Brian Nixon entwickelte und vorgestellte NFA-Framework [MyCN92] vertritt den prozessorientierten qualitativen Ansatz. Es ist aus der Motivation entstanden, Entwurfsentscheidungen auf der Basis von NFA zu treffen [MyCN92, S.1] und diese besser analysieren und dokumentieren zu können.
Von dem Zeitpunkt seiner Entstehung an wurde es konsequent weiterentwickelt [CNYM99] und von vielen Methoden und Ansätzen adaptiert. Um die zeitliche Entwicklung des Frameworks zu verdeutlichen und das Verständnis zu erleichtern, welche Teile des Frameworks von anderen Methoden verwendet werden, wird es zunächst in seiner nicht erweiterten Form dargestellt.
Der Begriff *Softgoal* wurde bei der Vorstellung des NFA-Frameworks noch nicht verwendet, um NFA zu beschreiben. Diese wurden einfach als Ziele (eng. Goals) bezeichnet. Da dieser Begriff jedoch später in vielen angrenzenden Bereichen Verwendung fand, erfolgte eine Namensänderung um einer eventuellen Verwechslungsgefahr vorzubeugen. Zudem sollten Ähnlichkeiten und Unterschied zwischen FA (*funktionale Ziele*) und NFA (Softgoals) innerhalb der zielorientierten Anforderungsanalyse verdeutlicht werden. Um eine einheitliche Begriffsnutzung und somit ein besseres Verständnis zu ermöglichen, wird in dieser Arbeit direkt

der Begriff des Softgoal definiert und verwendet. Gleiches gilt für den Begriff Softgoal-Interpendenz-Graph (*SIG*).
Da eine detaillierte Beschreibung des NFA-Frameworks über den Rahmen dieser Arbeit hinaus gehen würde, wird an dieser Stelle ausdrücklich darauf verwiesen, dass hier nur eine Einführung ohne Anspruch auf Vollständigkeit gegeben werden kann.

3.2 Grundlagen

Das NFA-Framework basiert auf der Evaluierung von NFA, die in einer ersten Analyse als für das System kritische Qualitätseigenschaften identifiziert wurden. Für diese Analyse wird auf verschiedene Arten von Softgoals zurückgegriffen, die in Abbildung 1 aufgeführt sind.

Bezeichnung	**Definition**	**Beispielanwendung im SIG**
NFA-Softgoals	Innerhalb des NFA-Frameworks werden NFA durch NFA-Softgoals repräsentiert.	Erreichbarkeit der Konten
Operative Softgoals	Ein operatives Softgoal ist eine konkrete Entwurfsentscheidung, um einem NFA-Softgoal zu entsprechen.	Benutzer identifizieren
Kritische Softgoals	Kritische Softgoals sind die NFA-Softgoals, auf deren Basis mit der Anwendung des NFA-Frameworks begonnen wird.	Sicherheit Konto

Abbildung 1. Arten von Softgoals (basiert auf [CNYM99, S.xxii, S.27])

Die formale Notation für einzelne NFA-Softgoals lautet „NfaTyp[Themengebiet]". Der erste Teil (NfaTyp) steht für die Art der NFA, wie beispielsweise Sicherheit oder Genauigkeit. Der zweite Teil (Themengebiet) beschreibt den Kontext. Die zweite Möglichkeit besteht darin, wie in den Beispielen dieser Arbeit, auf eine informelle Notation zurückzugreifen. Diese ist in der Regel intuitiver und schneller erfassbar.
Ein NFA-Softgoal gilt als komplett erfüllt, sobald die ihm untergeordneten Ziele erfüllt wurden [MyCY99, S.5]. Da es sich bei NFA, im Gegensatz zu FA, um Qualitätseigenschaften für ein gesamtes System handeln kann (siehe Kapitel 2.1), ist es häufig schwer festzustellen, wann diese komplett erreicht wurden. Dies gilt besonders, wenn sie nicht direkt quantitativ erfassbar sind. Daher gilt es vielmehr ein NFA-Softgoal in einem akzeptablen Rahmen zu erfüllen [MyCN92, S.3].
Für die Beschreibung der zur Genüge erfolgten Erfüllung eines *NFA-Softgoals*

wurde der englische Begriff *satisficing* [DaLF93] kreiert. Dieses Wort stellt eine Kombination der Begriffe satisfying (zufrieden stellend) und sufficing (ausreichend) dar. Dies erschwert das Finden einer angemessenen Übersetzung. Um die Lesbarkeit dieser Arbeit zu erhöhen, wird zur Darstellung dieses Sachverhalts der Term „*einem NFA-Softgoal entsprechen*" verwendet.

Definition: „*Einem NFA-Softgoal entsprechen*" ist die Übersetzung der englischen Wortschöpfung saticeficed. Er wird verwendet um zu verdeutlichen, dass ein NFA-Softgoal innerhalb der geforderten Maße erfüllt wurde [MyCY99, S.5].

NFA-Softgoals können verschieden Prioritäten (neutral, kritisch und sehr kritisch) besitzen und in weitere *NFA-Softgoals* aufgeteilt werden. Dies kann unter Anderem dazu dienen, NFA-Softgoals zu präzisieren oder zu verdeutlichen. *Operative Softgoals* können schließlich auf positive oder negative Weise dazu beitragen einen NFA-Softgoal zu entsprechen.

Wie eingangs erläutert, ist ein wichtiger Aspekt bei der Arbeit mit NFA die einfache Darstellung von Wechselwirkungen, die Verknüpfung von Anforderungen und Entwurfsentscheidungen und eine rückverfolgbare Dokumentation der Analyse- und Spezifikationsarbeit. Das NFA-Framework verfügt zu diesem Zweck über *Softgoal-Interpendenz-Graphen (SIG)*. An der Spitze einzelner SIGs stehen die identifizierten kritischen NFA-Softgoals. Einzelne NFA-Softgoals können durch verschiedene Arten von *Wechselwirkungskanten* verbunden werden.

Definition: *Wechselwirkungskanten* (eng. Interdependency links) sind (meist gerichtete) Kanten in SIGs. Durch sie werden die verschiedenen Arten von Wechselwirkungen zwischen einzelnen operativen Softgoals und NFA-Softgoals ausgedrückt. Diese Wechselwirkungen können auf positiv, negativ oder gar nicht dazu beitragen einem Softgoal zu entsprechen. [CNYM99, S.29]

Der gewöhnliche Umgang mit NFA und ihre Einbeziehung in den Entwicklungsprozess ist unstrukturiert. Im Gegensatz dazu existiert für viele Problemstellungen bezüglich FA zum Beispiel eine Menge von Entwurfsmustern, die eine Vielzahl von Vorteilen bieten. Um Analyse- und Entwurfswissen wiederzuverwenden, ein schnelleres Vorgehen zu ermöglichen und den Qualitätsstandard zu erhöhen, sollten daher auch für die Arbeit mit NFA Entwurfskataloge verwendet werden [CNYM99, S.17]. Das NFA-Framework empfiehlt dabei die Strukturierung in drei verschiedene Arten von Katalogen. Erstens wird erworbenes Wissen bezüglich einzelner Typen von NFA wie beispielsweise Performanz gesammelt. Zweitens können vorteilhafte Entwicklungstechniken, die sich zur Erfüllung von bestimmten Anforderungen bewährt haben katalogisiert werden. Schließlich dient ein Katalog zum Beschreiben der unterschiedlichen Auswirkungen von Entwurfsentscheidungen auf verschiedenen NFA [CNYM99, S.17].

3.3 Anwendung

Die folgende Beschreibung der Anwendung des *NFA-Frameworks* basiert auf den folgenden Quellen [MyCN92,CNYM99,SuCh01]. Die von den verschiedenen Anspruchsberechtigten identifizierten kritischen Softgoals bilden den Ausgangspunkt für die Konstruktion der zu erzeugenden SIGs. In dem aufgeführten Beispiel (siehe Abbildung 2) sind dies die kritischen Softgoals „gute Performanz" und „Sicherheit".
Im nächsten Schritt können diese noch sehr wage definierten kritischen Softgoals weiter verfeinert werden. Dies kann mit der Hilfe von verschiedenen Methoden geschehen, deren Verwendung davon abhängt, auf welche Art und Weise die untergeordneten NFA-Softgoals dazu beitragen, dem ihnen übergeordneten zu entsprechen.
Dabei ist es bei einer **UND** Aufspaltung nötig, allen untergeordneten NFA-Softgoals zu entsprechen, um dies auch für das übergeordnete NFA-Softgoal zu ermöglichen (siehe Abbildung 2). Eine **ODER** Aufspaltung stellt dabei Alternativen dar. Es genügt daher einem untergeordneten NFA-Softgoal zu entsprechen. Um diesen Vorgang zu erleichtern, sollte dabei auf das Wissen aus Entwurfskatalogen zurückgegriffen werden. Graphisch wird eine UND-Methode durch einen einfachen Bogen im SIG gekennzeichnet (siehe Abbildung 2). Für ODER-Methoden wird auf einen doppelten Bogen zurückgegriffen.

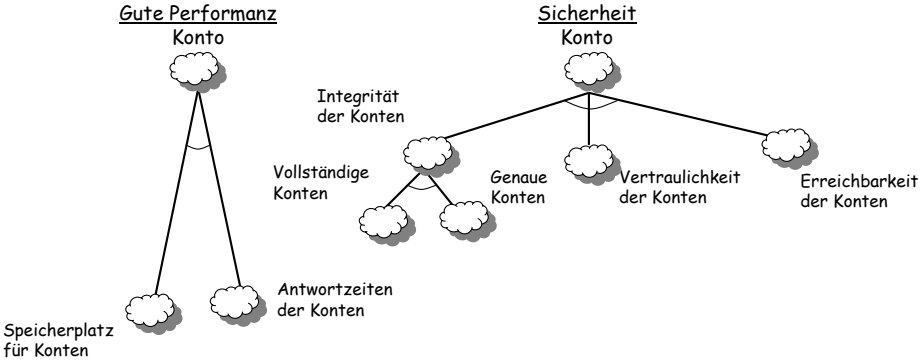


Abbildung 2. Verfeinerung von Softgoals (ähnlich in [CNYM99, S.17])

Wurden die verschiedenen NFA-Softgoals zur Genüge verfeinert, wird eine Menge von operativen Softgoals gesucht, um einzelnen NFA-Softgoals zu entsprechen. Durch das Ergänzen von Wechselwirkungskanten wird angezeigt, wie ein operatives Softgoal dazu beiträgt einem NFA-Softgoal zu entsprechen. Es ist auch möglich, dass sich ein operatives Softgoal auf verschiedene NFA-Softgoals auswirken kann.
Nun können die Wechselwirkungen zwischen den unterschiedlichen NFA-Softgoals

analysiert und abgewogen werden. Als Ergebnis dieser Tätigkeit wird eine Menge von operativen Softgoals ausgewählt. Dabei werden ausgewählte operative Softgoals mit einem Haken im SIG gekennzeichnet. Anhand von Kennzeichnungsfunktionen werden dann die einzelnen SIGs von den operativen Softgoals ausgehend ausgewertet und für jedes NFA-Softgoal bestimmt, ob ihm entsprochen wird oder nicht (siehe Abbildung 3).
Wurde den gewünschten NFA-Softgoals entsprochen, kann auf der Basis der ausgewählten operativen Softgoals mit dem funktionalen Entwurf und der Analyse des Softwaresystems fortgefahren werden.

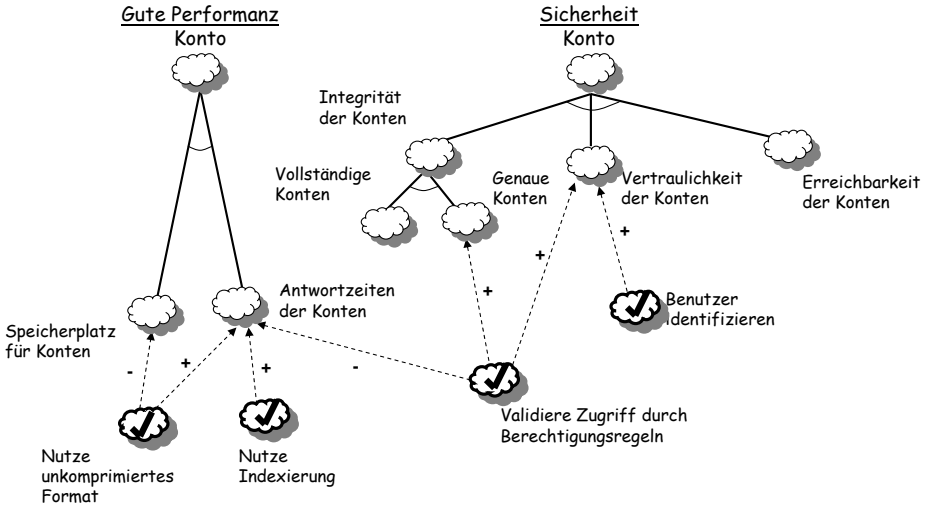


Abbildung 3. Auswahl der operativen Softgoals (ähnlich in [CNYM99, S.36])

3.4 Bewertung

In seiner Grundform bietet das NFA-Framework bereits eine Vielzahl von Vorteilen und nimmt sich einiger der eingangs beschriebenen Probleme an. Es ermöglicht den zuvor unstrukturierten Umgang mit NFA gezielt durch ein ingenieurstechnisches, festgelegtes Vorgehen zu ersetzen. So wird es möglich frühzeitig gewisse Vorhersagen über die Qualität des Endprodukts und die Einhaltung der identifizierten NFA zu treffen. Dadurch erwachsen unter anderem wirtschaftliche Vorteile. Nicht den Anforderungen entsprechende Softwareprodukte müssen nicht unnötig weit entwickelt oder konkret implementiert werden. NFA können leichter verifiziert werden.
Des Weiteren werden die unterschiedlichen Beeinflussungen von verschiedenen NFA durch bestimmte Entwurfsentscheidungen verdeutlicht und können gezielt

abgewogen genommen werden. Verschiedene Alternativen um NFA zu entsprechen sind ersichtlicht. Diese Punkte ermöglichen es, gezielte Trade-Off- und Entwurfsentscheidungen zu treffen, durch die die Qualität des Endprodukts verbessert werden kann.
Außerdem ist es zu einem gewissen Teil möglich, zu ermitteln, welche Entwurfsentscheidung aufgrund von NFA getroffen wurde. Dies verbessert die Rückverfolgbarkeit.
Neben diesen beachtlichen Vorteilen sind allerdings auch einige Kritikpunkt und Verbesserungsmöglichkeiten erkennbar. Die Integration in den Softwareentwicklungsprozess wird nicht eindeutig beschrieben. Somit ist zunächst fraglich, wie eine vollständige Basismenge von kritischen NFA ermittelt werden soll. Der noch nicht ausgereifte und zu oberflächliche Erhebungsprozess wird von Cysneiros kritisiert [CydPL04, S.1].
Auf der anderen Seite wird kein genaues Vorgehen beschrieben, wie und wann auf der Basis von NFA getroffene Entwurfentscheidungen in einen konkreten Entwurf integriert werden sollen.
Bei der Verfeinerung von NFA-Softgoals und bei der Suche nach operativen Softgoals kann auf katalogisiertes Wissen und standardisierte Methoden zurückgegriffen werden. Dies ermöglicht, das Vorgehen zu einem gewissen Grad zu automatisieren. Trotz allem stellt bei größeren Problemstellungen die Skalierung und praktische Anwendbarkeit einen Problempunkt dar, da manuelle Bewertungsmethoden verwendet werden.
Zudem ist, bei einer geringen Wissenbasis, fraglich ob bei der Auswahl von alternativen operativen Softgoals auch die für die konkrete Problemstellung passende Alternative aufgeführt wird. Bei der manuellen Analyse der SIGs und der Auswahl einer Menge von operativen Softgoals wäre zudem eine Unterstützung zum Auffinden von optimalen Entwurfsentscheidungen wünschenswert.
Vorhandene Ansatzpunkte für Verbesserungen und seine zahlreichen Vorteile führten dazu, dass das NFA-Framework weiterentwickelt und in verschiedene Anforderungsanalyseverfahren integriert wurde [CyLe01,ChNi95,XZRA05].
Zudem bleiben die Fragen bestehen, welchen genauen Stellenwert die Analyse von FA in Verbindung mit dem NFA-Framework annimmt. Werden FA getrennt von NFA analysiert? Sind beide Arten von Anforderungen gleichgestellt? Wie werden bei getrennten Sichten asynchrone Entwicklungen und Inkonsistenzen zwischen FA und NFA verhindert [CydPL04, S.2]? Eine Möglichkeit zur gemeinsamen Analyse von NFA und FA bietet die Nutzung des NFA-Frameworks in der zielorientierten Anforderungsanalyse.

4 Zielorientierte Anforderungsanalyse

4.1 Von prozessorientiert zu zielorientiert

„Die Komplexität eines Informationssystems wird teilweise von seiner Funktionalität (FA) [...] und teilweise von übergeordneten Anforderungen für seine Entwicklung (NFA) [...] bestimmt" [MyCN92, S.1]. Daher sollte bei der Anforderungsanalyse keiner der beiden Aspekte vernachlässigt werden. Der Nutzen des

kompletten Endproduktes kann nicht durch die isolierte Betrachtung nur eines Anforderungstyps bereits frühzeitig ermittelt werden. Daraus folgt, dass bei der Bewertung von Alternativen in der Anforderungsanalyse allgemeine Ziele der verschiedenen Anspruchsberechtigten als Kriterium verwendet werden sollten [MCLW+01, S.1]. Aus diesem Ansatz heraus erfolgte die Weiterentwicklung des NFA-Frameworks zum zielorientierten Analysewerkzeug. In der Literatur sind neben der hier vorgestellten, noch eine Vielzahl von anderen zielorientierten Anforderungsanalysemethoden vertreten.

4.2 Grundlagen

Bei der zielorientierten Anforderungsanalyse werden zunächst FA des zu erstellenden Softwareprodukts von den verschiedenen Anspruchsberechtigte identifiziert [YKCB06, S.1].

Definition: Als *funktionale Ziele* werden FA in der zielorientierten Anforderungsanalyse bezeichnet. Sie werden komplementär zu Softgoals betrachtet und stellen zu erfüllenden Aufgaben dar [MCLW+01, S.2].

Funktionale Ziele können wie NFA-Softgoals durch UND/ODER-Methoden verfeinert werden. Dadurch können sie zum einen hierarchisch in Unterziele aufgegliedert werden. Auf der anderen Seite wird es ermöglicht, Alternativen zum Erreichen eines Ziels darzustellen. Ein übergeordnetes funktionales Ziel gilt als erfüllt, sobald alle (UND) oder mindestens eins (ODER) der ihm untergeordneten funktionalen Ziele erfüllt wurden. UND-Methoden werden graphisch durch einen einfachen, ODER-Methoden durch einen doppelten Bogen dargestellt [MCLW+01, S.2]. Durch die Anwendung der UND/ODER-Methoden entsteht eine baumartige Graphenstruktur.

Definition: Als ein *Blattziel* bezeichnet man die funktionalen Ziele, die Blätter in der Struktur darstellen, die durch die Anwendung von UND/ODER-Methoden auf ein funktionales Ziel entstehen.

Die Ergebnisse des Modellierungs- und Analyseprozesses kann in Graphenform dokumentiert werden. Diese ähneln den SIGs des NFA-Frameworks.

4.3 Anwendung

Die Informationen für die Beschreibung der vorgestellten zielorientierten Analysemethode wurden den folgenden Quellen entnommen: [MyCY99,MCLW+01]. Es wird wiederum kein Anspruch auf Vollständigkeit erhoben.
Zu Beginn des Analyseprozesses wird eine Menge von identifizierten FA und NFA des zu erstellenden Softwareprodukts benötigt. Die FA stellen die zu erfüllenden funktionalen Ziele dar. Wie in Abbildung 4 zu erkennen, existiert in unserem Beispiel ein funktionales Ziel „Termine planen". Dieses wird im Folgenden zunächst durch die Anwendung einer UND-Methode in „Ermittle noch freie

Termine" und „Wähle Terminplan" aufgeteilt. Die Anwendung der UND/ODER-Methoden kann nach belieben fortgesetzt werden.

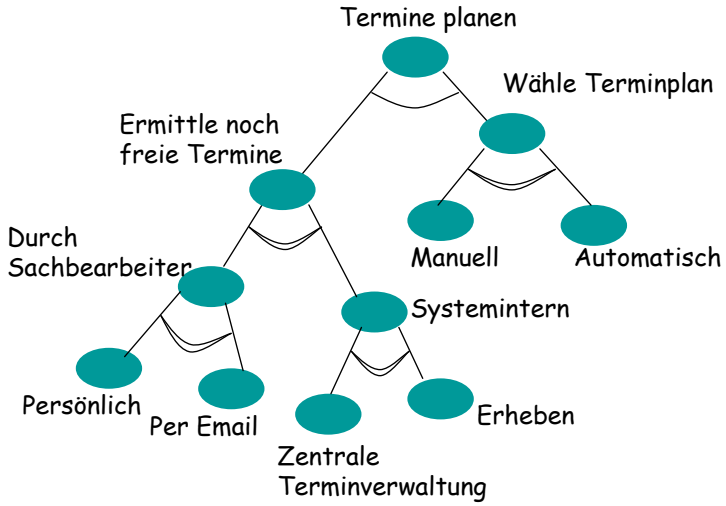


Abbildung 4. Verfeinerung funktionaler Ziele (ähnlich in [MCLW+01, S.2])

Im nächsten Schritt werden die NFA mit Hilfe von SIGs analysiert. Hierbei wird auf die vorgestellten Techniken des NFA-Frameworks zurückgegriffen (siehe Kapitel 3). Dabei ist wiederum auf unterschiedliche Wechselwirkungen zu achten.

Darauf hin werden die bis dahin getrennt entwickelten hierarchischen Modelle für funktionale Ziele und NFA-Softgoals zusammengeführt und wiederum auf Wechselwirkungen untersucht (siehe Abbildung 5). Jeder (Alternativ-)Teil eines funktionalen Ziels kann sich dabei positiv, negativ oder gar nicht darauf auswirken, einzelnen NFA-Softgoals zu entsprechen. Dadurch können wiederum Konflikte entstehen. In unserem Beispiel wird dies dadurch verdeutlicht, das eine automatische Auswahl des Terminplans zur Qualität des erstellten Planes beiträgt, indem Konflikte minimiert werden. Gleichzeitig wirkt es sich jedoch negativ auf den Mitwirkungsgrad - eine Anpassung an individuelle Bedürfnisse - und damit wiederum auf die Qualität aus. Es ist daher möglich, dass nicht allen NFA-Softgoals entsprochen werden kann.
Schließlich wird eine Menge von Blattzielen ausgewählt, die alle anfangs identifizierten funktionalen Ziele erfüllen (siehe Abbildung 5). Dabei sollte auch darauf geachtet werden, den gewünschten NFA-Softgoals zu entsprechen. Diese Blattziele repräsentieren die in der weiteren Entwicklung zu berücksichtigenden Entwurfsentscheidungen.

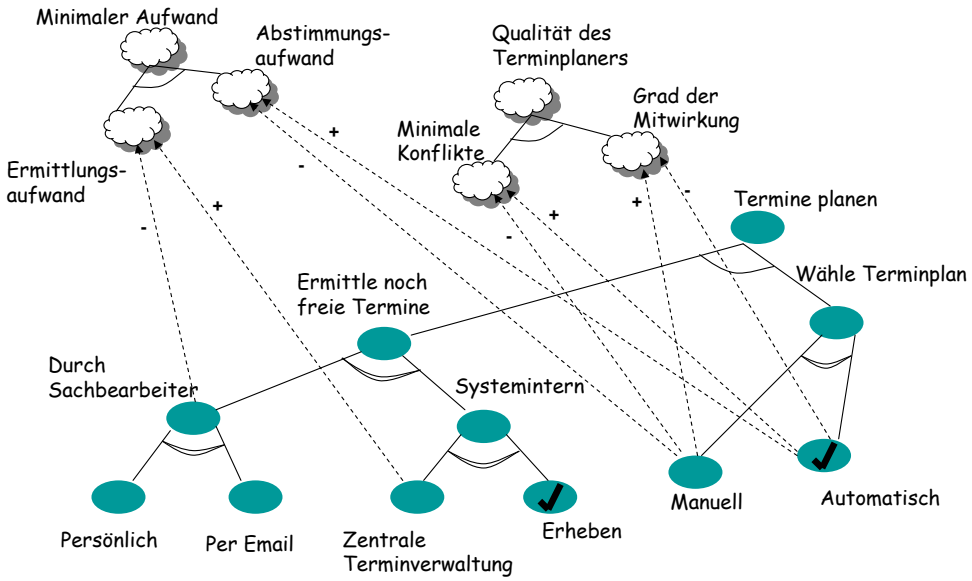


Abbildung 5. Integration von funktionalen Zielen und Softgoals (ähnlich in [MCLW+01, S.4])

4.4 Bewertung

Die Nutzung des NFA-Frameworks in Verbindung mit der zielorientierte Anforderungsanalyse verbessert die Anforderungsspezifikation an verschiedenen Punkten. Durch die gemeinsame Betrachtung von FA und NFA wird die interne Konsistenz verbessert, da Konflikte zwischen beiden Anforderungsarten besser entdeckt werden können. Die differenzierte Betrachtung von funktionalen Zielen fördert die Rückverfolgbarkeit und Eindeutigkeit, da einzelne Anforderungen in einen bestimmten Kontext gesetzt werden. Die Korrektheit profitiert durch die Modellierung von Alternativen für das Erreichen von funktionalen Zielen. Es kann besser nachvollzogen werden, dass eine Entwurfsentscheidung nur aufgrund von NFA getroffen wurde und daher eine Darseinberechtigung besitzt.
Noch nicht vorhanden sind (automatisierte) Methoden und Bewertungskriterien zum Finden von optimalen Lösungsmengen. Dadurch bleibt weiterhin die Skalierbarkeit der Methode fraglich. Unklar ist auch, wie sich die vorgestellte zielorientierte Analyse in den Softwareentwicklungsprozess eingliedert. Wie genau wird die benötigte Grundmenge von FA und NFA zuverlässig und vollständig erhoben? Mit welchem genauen Vorgehen können die gewonnenen Ergebnisse, in einen z.B. mit der UML (Unified Modeling Language) modellierten Entwurf, integriert werden? Eine mögliche Lösung für diese Fragestellungen bietet der im Folgenden dargestellte LEL-Ansatz.

5 Der LEL-Ansatz

Durch die Entwicklung des NFA-Frameworks wurden unter anderem viele theoretische Grundlagen für den Umgang mit NFA gelegt. Für den Einsatz in der Praxis ist es allerdings nötig das NFA-Framework besser in den Softwareentwicklungsprozess zu integrieren. Einen weit verbreiteten Ansatz stellt dabei die Nutzung eines Erweiterten-Sprachlexikons (eng. Language Extended Lexicon, *LEL*) [LeFr93a] zur Erhebung von FA und NFA dar. Daraufhin findet das NFA-Framework Anwendung, mit der anschließenden Integration der Ergebnisse in einen per UML modellierten konzeptuellen Entwurf.
Der LEL-Ansatz wurde von Luiz Marcio Cysneiros und Julio Cesar Sampaio do Prado Leite entwickelt und ist detailliert in den folgenden Quellen beschrieben: [CyLe01,CydPL04,LeFr93a].

5.1 LEL

Der Zweck des *LEL* ist die Aufstellung eines Vokabulars, dass die Anwendungsdomäne (eng. Universe of Discourse, *UofD*) beschreibt. Dieses sollte sowohl funktionale als auch nichtfunktionale Eigenschaften der Software enthalten. Es basiert darauf, „die Sprache eines Problems zu verstehen, ohne [...] das Problem selbst zu verstehen" [LeFr93b, S.1].

Definition: "Das *UofD* ist der generelle Kontext indem eine Software entwickelt werden und arbeiten sollte. Das UofD beinhaltet alle Informationsquellen und alle bekannten Personen die in Beziehung zu der Software stehen. Diese Personen sind auch als Akteure in diesem UofD bekannt" [CyLe01, S.2].

Ein LEL besteht aus einer Menge von Einträgen. Jeder Eintrag wird wiederum durch *Absichten* und *Antwortverhaltensweisen* beschrieben.

Definition: *Absichten* (eng. Notions) innerhalb eines LEL-Eintrags beschreiben dessen Bedeutung und grundlegenden Eigenschaften [CyLe01, S.3].

Definiton: Das *Antwortverhalten* (eng. Behavioral Responses) innerhalb eines LEL-Eintrags spezifiziert dessen Assoziationen zu anderen Einträgen in einem UofD [CyLe01, S.3].

Die Erstellung des LEL orientiert sich an den zwei grundlegenden Prinzipien des minimalen Vokabulars und der Zyklenhaftigkeit. Ersteres besagt, dass bei der Beschreibung eines Eintrags nicht auf den LEL unbekannte Einträge zurückgegriffen werden sollte. Das zweite Prinzip fordert einzelne Einträge durch möglichst viele Referenzen auf andere Einträge zu beschreiben. Diese beiden Prinzipien führen dazu, dass das LEL eine möglichst in sich geschlossene Spezifikation bildet.
Stellen wir uns für ein Beispiel ein UofD vor, dass mit Hilfe eines LELs beschrieben werden soll. Dabei besteht das UofD nur aus einem Quadrat. Der erste

Eintrag „Quadrat" wird durch die Absicht „hat Kantenlänge" und das Antwortverhalten „besitzt vier Kanten" beschrieben. Gemäß dem Prinzip des minimalen Vokabulars, wird als nächstes der Eintrag „Kante" in unserem UofD ergänzt. Dieser erhält die Absicht „hat Länge" und nach dem Prinzip der Zyklenhaftigkeit als Antwortverhalten einen Rückverweis auf den Eintrag Quadrat „kann Teil eines Quadrats sein".
Generell dient das LEL als Wissenbasis für den weiteren Entwicklungsprozess, in dem wiederum das NFA-Framework Anwendung findet.

5.2 Anpassung des NFA Frameworks

Für die Verwendung im LEL-Ansatz wird das NFA-Framework leicht angepasst. Daher wird für die graphische Präsentation der Ergebnisse der Begriff *NFA-Graph* anstelle von SIG verwendet. Für die Erstellung der einzelnen NFA-Graphen wird jeder Eintrag im LEL nach NFA durchsucht. Für jede gefundene NFA wird daraufhin ein neuer NFA-Graph konstruiert. Dabei werden, anders als im vorgestellten NFA-Framework, operative Softgoals in die zwei unterschiedlichen Klassen, *statische* und *dynamische operative Softgoals*, eingeteilt.

Bezeichnung	**Definition**	**Beispielanwendung NFA-Graph**
Statische operative Softgoals…	…drücken aus, dass es notwendig ist, bestimmte Daten zu speichern, um einem NFA-Softgoal zu entsprechen.	Maximale Nutzer = 1000
Dynamische operative Softgoals…	…stehen für abstrakte Konzepte und fordern eine auszuführende Handlung, um einem NFA-Softgoal zu entsprechen.	Benutzer identifizieren

Abbildung 6. Klassen der operativen Softgoals (basiert auf [MCLW+01, S.4])

Für die Notation der einzelnen NFA-Softgoals und operativen Softgoals wird wiederum so oft wie möglich auf Einträge des verwendeten LEL zurückgegriffen oder neue Einträge ergänzt.

5.3 Anwendung

Wie in Abbildung 7 zu erkennen basiert der *LEL*-Ansatz auf der getrennten Entwicklung und Analyse von FA und NFA in einer funktionalen Sicht und nichtfunktionalen Sicht, die an festgelegten Punkten ineinander integriert werden. Innerhalb der funktionalen Sicht wird dabei auf die Modellierungsmöglichkeiten

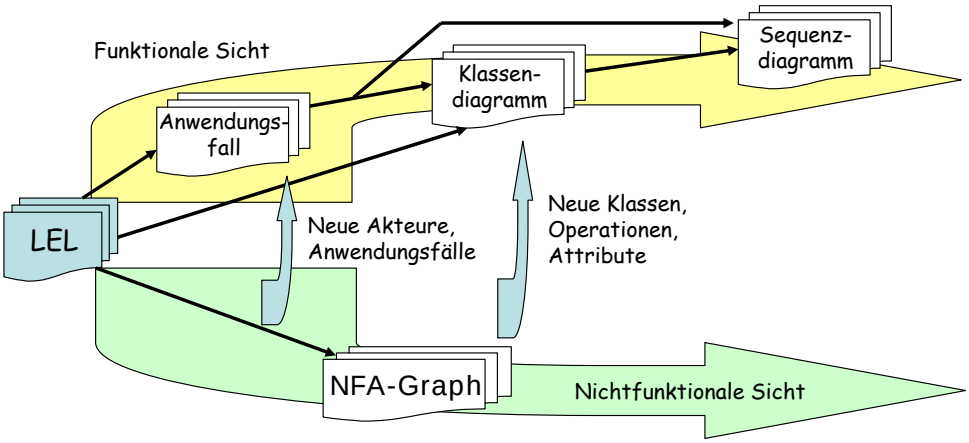


Abbildung 7. Vorgehen im LEL Ansatz (ähnlich in [CyLe01, S.2])

der UML zurückgegriffen. Die nichtfunktionale Sicht wird mit Hilfe des NFA-Frameworks erarbeitet.
Die zwei großen Blockpfeile visualisieren dabei den fortlaufenden Entwicklungsprozess. Zu verschiedenen Zeitpunkten können die Ergebnisse der nichtfunktionalen- in die funktionale Sicht integriert werden. Dies kann dazu führen, dass verschiedene Diagramme der funktionalen Sicht um neue Elemente erweitert werden. Den Ausgangspunkt für diesen Prozess stellt das LEL dar.

Um die Ergebnisse der nichtfunktionalen Sicht in die funktionale Sicht zu integrieren existieren verschiedene Vorgehensweisen. Da eine umfangreiche Beschreibung der verschiedenen Techniken über den Rahmen dieser Arbeit hinausgehen würde, wird hier nur die Integration in per UML modellierte Anwendungsfalldiagramme erläutert.
Zuerst werden die einzelnen erstellten Anwendungsfalldiagramme auf Wörter durchsucht, die als Einträge im LEL vorhanden sind. Dies gilt insbesondere für die Diagrammnamen, einzelne Anwendungsfälle und Akteure. In Abbildung 8a wurde beispielsweise der Eintrag „Benutzer" identifiziert.
Daraufhin werden die einzelnen NFA-Graphen untersucht, ob sie den entsprechenden Eintrag enthalten. In unserem Beispiel ist dies in einem NFA-Graphen der Fall, der sich mit der NFA Sicherheit befasst (siehe Abbildung 8b und vergleiche Kapitel 3.3). Dabei ist es möglich, dass ein Eintrag in mehreren NFA Graphen auftritt.
Schließlich wird in dem korrespondierenden Anwendungsfalldiagramm überprüft, ob es die geforderten dynamischen operativen Softgoals berücksichtigen. Daher ob es durch einen Anwendungsfall realisiert wird. Ist dies nicht der Fall, wird das ursprüngliche Diagramm und ein entsprechender Anwendungsfall ergänzt (siehe Abbildung 8c.
Sollte in einem Anwendungsfalldiagramm kein Eintrag des LEL gefunden wer-

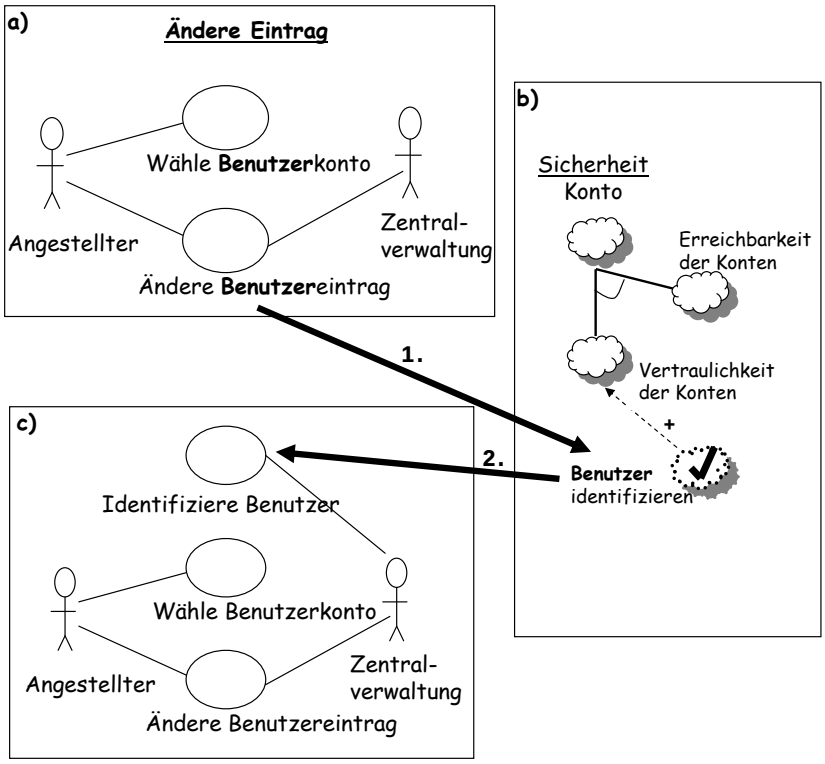


Abbildung 8. Integration der nichfunktionaler Sicht in ein Anwendungsfalldiagramm (ähnlich in [CydPL04, S.11])

den, sollte es auf Vollständigkeit und Eindeutigkeit überprüft werden. Im ersten Fall könnten wichtige Informationen des UofD nicht durch LEL erfasst worden sein. Letzteres könnte auf die selbe Bezeichnung für zwei unterschiedliche Einträge hinweisen. Daher fördert die Integration beider Sichten auch die Qualität der genutzten Wissengrundlage.

5.4 Bewertung

Der LEL-Ansatz stellt eine sinnvolle Methode zur Integration des NFA-Frameworks in den Softwareentwicklungsprozess dar. Er verbessert und strukturiert durch die Verwendung des LEL die Wissensbasis, die für die Verwendung des NFA-Frameworks benötigt wird. Die getrennte Entwicklung von funktionaler und nichtfunktionaler Sicht mit regelmäßiger Integration ineinander, fördert die Vollständigkeit, Validierbarkeit und Konsistenz der Anforderungsspezifikation und darauf basierenden Entwürfen.
Zudem fördert der Ansatz die praktische Anwendbarkeit des NFA-Frameworks, indem er einen Verbindungspunkt zur objektorientierten Analyse vorstellt. Diese stellt mit ihren ausgereiften Werkzeugen und Modellierungsmöglichkeiten, beispielsweise mit Hilfe der UML, eine der am meisten angewendeten Methoden für den Umgang mit FA dar.
In der Literatur ist allerdings neben dem LEL-Ansatz eine Vielzahl von weiteren Ansätzen vertreten, die das gleiche Ziel verfolgen. Darunter der Zielorientierte-Anwendungsfall-Ansatz [SuCh05], das Unifying Framework [Kava02] oder die nichtfunktionale Dekomposition [PodW04]. Auf der anderen Seite kann alternativ für die anfängliche Erhebung der Wissensbasis, auf Ansätze wie den Volere-Anforderungs-Prozess [RoRo99], zurückgegriffen werden.

6 Fazit und Ausblick

Die frühzeitige Risikoabschätzung und -minimierung ist einer der Hauptmotivationspunkte im Requirements Engineering. NFA spielen dabei eine bedeutende Rolle, ob ein fertiges Softwareprodukt von einem Kunden akzeptiert wird. Das NFA-Framework ermöglicht, als prozessorientierter Ansatz, eine frühzeitige Analyse und formale Betrachtung von NFA. Dadurch kann maßgeblich die Qualität von Anforderungsspezifikationen und fertigen Softwareprodukten verbessert werden.
Seit der Vorstellung seiner theoretischen Grundlagen, die sich auf den Umgang mit NFA beschränkten, wurde das NFA-Framework schnell für umfangreichere Ansätze in der Anforderungstechnik entdeckt. Darunter beispielsweise die zielorientierte Anforderungsanalyse, die in Kombination mit der objektorientierten Analyse verwendet werden kann. Für die praktische Anwendung des NFA-Frameworks ist eine fließende Integration in den Softwareentwicklungsprozess nötig. In der Literatur sind dafür parallel eine Vielzahl von unterschiedlichen Ansätzen entwickelt worden. Diese zeichnen sich oft durch einen modularen Aufbau aus. Diese Arbeit soll diesen Aufbau verdeutlichen und das Verständnis und die

Bewertung von unterschiedlichen Methoden erleichtern. Für eine konkrete Anwendung des NFA-Frameworks sollte daher, abhängig von der Problemstellung, eine Kombination von unterschiedlichen Ansätzen in Erwägung gezogen werden. Damit das NFA-Framework breite Anwendung in der Praxis finden kann, ist noch immer eine bessere Integration notwendig. Dabei spielt besonders die Automatisierung von einzelnen Vorgängen eine bedeutende Rolle. Innerhalb des NFA-Frameworks sind dies unter anderem bessere Bewertungsmethoden für Alternativen und das Finden von optimalen Lösungsmengen. Außerdem sollen zukünftig Modelle für FA und NFA besser in das Design von Softwarearchitekturen einbezogen werden [SuCh05, S.8].

Literatur

ChNi95. Lawrence Chung und Brian A. Nixon. Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach. In *International Conference on Software Engineering*, 1995, S. 25–37.

CNYM99. Lawrence Chung, Brian A. Nixon, Eric Yu und John Mylopoulos. *Non-Functional Requirements in Software Engineering (THE KLUWER INTERNATIONAL SERIES IN SOFTWARE ENGINEERING Volume 5) (International Series in Software Engineering)*. Springer. October 1999.

CydPL02. L.M. Cysneiros und J.C.S. do Prado Leite. Non-functional requirements: from elicitation to modelling languages. *Proceedings of the 24rd International Conference on Software Engineering, 2002. ICSE 2002.*, 2002, S. 699–700.

CydPL04. L.M. Cysneiros und J.C.S. do Prado Leite. Nonfunctional requirements: from elicitation to conceptual models. *IEEE Transactions on Software Engineering*, 30(5), May 2004, S. 328–350.

CyLe01. Luiz Marcio Cysneiros und J.C.S.do.P. Leite. Using UML to reflect non-functional requirements. In *CASCON '01: Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2001, S. 2.

DaLF93. Anne Dardenne, Axel van Lamsweerde und Stephen Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20(1-2), 1993, S. 3–50.

ErYM06. N.A. Ernst, Yijun Yu und J. Mylopoulos. Visualizing non-functional requirements. *Requirements Engineering Visualization, 2006. REV '06.*, Sept. 2006, S. 2–2.

Fran98. X. Franch. Systematic formulation of non-functional characteristics of software. *Third International Conference on Requirements Engineering, 1998. Proceedings.*, 6-10 1998, S. 174–181.

Glin06. Martin Glinz. Requirements Engineering I. `http://www.ifi.uzh.ch/rerg/fileadmin/downloads/teaching/courses/requirements_engineering_I_ws0607/Kapitel_01_Grundl.pdf`, Universität Zürich, Institut für Informatik, 2006. [Online, accessed 10-Juli-2008].

Kava02. Evangelia Kavakli. Goal-Oriented Requirements Engineering: A Unifying Framework. *Requir. Eng.*, 6(4), 2002, S. 237–251.

LeFr93a. J.C.S.do.P. Leite und A.P.M. Franco. A strategy for conceptual model acquisition. *Proceedings of IEEE International Symposium on Requirements Engineering, 1993.*, Jan 1993, S. 243–246.

LeFr93b. J.C.S.do.P. Leite und A.P.M. Franco. A strategy for conceptual model acquisition. *Proceedings of IEEE International Symposium on Requirements Engineering*, Jan 1993, S. 243–246.

MCLW+01. John Mylopoulos, Lawrence Chung, Stephen Liao, Huaiqing Wang und Eric Yu. Exploring Alternatives During Requirements Analysis. *IEEE Software*, 18(1), 2001, S. 92–96.

MyCN92. J. Mylopoulos, L. Chung und B. Nixon. Representing and using nonfunctional requirements: a process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6), Jun 1992, S. 483–497.

MyCY99. John Mylopoulos, Lawrence Chung und Eric Yu. From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42(1), 1999, S. 31–37.

oEEn98. The Institute of Electronical und Electronics Engineers. IEEE recommended practice for software requirements specifications. *IEEE Std 830-1998*, Band Revision of IEEE Std 830-1993, 20 Oct 1998.

PeRD00. Don Peppers, Martha Rogers und Robert Dorf. *The one to one fieldbook*. Capstone. Repr.. Auflage, 2000.

PodW04. E.R. Poort und P.H.N. de With. Resolving requirement conflicts through non-functional decomposition. *Fourth Working IEEE/IFIP Conference on Software Architecture, 2004. WICSA 2004. Proceedings.*, 12-15 June 2004, S. 145–154.

Pohl08. Klaus Pohl. *Requirements Engineering*. dpunkt. 2008.

RoRo99. Suzanne Robertson und James Robertson. *Mastering the requirements process*. Addison-Wesley. 1999.

SuCh01. N. Subramanian und L. Chung. Software Architecture Adaptability – An NFR Approach, 2001.

SuCh05. S. Supakkul und L. Chung. A UML profile for goal-oriented and use case-driven representation of NFRs and FRs. *Third ACIS International Conference on Software Engineering Research, Management and Applications, 2005*, 11-13 Aug. 2005, S. 112–119.

XZRA05. Lihua Xu, Hadar Ziv, Debra Richardson und Thomas A. Alspaugh. An architectural pattern for non-functional dependability requirements. In *WADS '05: Proceedings of the 2005 workshop on Architecting dependable systems*, New York, NY, USA, 2005. ACM, S. 1–6.

YKCB06. Shuichiro Yamamoto, Haruhiko Kaiya, Karl Cox und Steven Bleistein. Goal Oriented Requirements Engineering: Trends and Issues. *IEICE - Trans. Inf. Syst.*, E89-D(11), 2006, S. 2701–2711.

Ressourcen-Dimensionierung

Jochen Maier

Betreuer: Michael Kuperberg

Zusammenfassung Eine fehlende Ressourcen-Planung in frühen Phasen der Software-Entwicklung kann im weiteren Projektverlauf zu schwerwiegenden Design-Fehlern führen, die Performance-Probleme nach sich ziehen. Die frühzeitige Planung der Hardware-Ressourcen, die benötigt werden um gewünschten Qualitätsmaßstäben und nicht-funktionalen Anforderungen gerecht zu werden, ist daher essentiell für einen erfolgreichen Projektabschluss. Dieser Prozess wird *Ressourcen-Dimensionierung* genannt. Im Rahmen der Arbeit werden verschiedene Methoden vorgestellt, mit denen die Performance von Ressourcen abgeschätzt werden kann, ohne dass das System bereits real implementiert sein muss. Teure Prototypen, um eine Messung durchführen zu können, bleiben so erspart. Ein Beispiel der Modellierung von Höchstleistungsrechnern am Ende der Ausführungen zeigt, wie diese Methoden in der Praxis eingesetzt werden können.

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit dem Thema der *Ressourcen-Dimensionierung*, also der Planung von performance-relevanten Hardware-Ressourcen, die benötigt werden, um ein Software-Projekt gemäß der gestellten Anforderungen zu einem erfolgreichen Abschluss zu bringen.

Dieser Prozess stellt ein für große Software-Projekte unabdingbar scheinendes Konzept dar. Durch die Integration der Überprüfung von Performance-Zielen in den gesamten Entwicklungs-Zyklus werden Qualitätsanforderungen so langfristig sichergestellt. Wirtschaftlichkeit und Auslastungseffizenz können frühzeitig analysiert und gegebenenfalls optimiert werden. Dazu werden geplante Systeme bezüglich ihres Lastverhaltens modelliert und Performance-Größen, die im Weiteren noch vorzustellen sind, abgeleitet. Dies stellt eine deutliche Verbesserung der Genauigkeit gegenüber oftmals eingesetzten Schätzmethoden dar. Durch eine exakte Vorhersage des System-Verhaltens können so Engpässe vorhergesehen und Überlastungen des Systems vermieden werden.

Die folgenden Themengebiete sind mit der Ressourcen-Dimensionierung eng verwandt und teilweise auch darin inbegriffen [Beck02, S. 182]. An dieser Stelle soll jedoch eine kurze Beschreibung als Einführung genügen und auf weiterführende Literatur verwiesen werden.

Kapazitätsplanung, Lastvorhersage [LaCh87,Sera86]
Analyse der aktuellen Arbeitslast, Vorhersage künftiger Veränderungen und der Größe der dafür benötigten System-Komponenten.

Hierbei wird jedoch davon ausgegangen, dass detaillierte Leistungsmessungen einer System-Umgebung bereits vorliegen und von diesen Werten ausgehend auf das neue System geschlossen werden kann. Die Ressourcen-Dimensionierung hingegen basiert auf abstrakten Größen und unvollständiger Information über die spätere System-Umgebung [WaMR04, S. 136].

Leistungsmessung, -analyse und -bewertung [Ferr78,BoRi89,HaZo95]
Untersuchung von vorliegenden Systemen hinsichtlich ihrer Leistungsgrößen.

System-Modellierung und -Optimierung [MeAD94,MeAD04]
Schematische Darstellung von geplanten oder bestehenden Rechner-Systemen.

1.1 Notwendigkeit der Ressourcen-Dimensionierung

Ein Blick in die Praxis zeigt, dass mangelnde Planung der Ressourcen und das Vernachlässigen frühzeitiger und wichtiger Performance-Entscheidungen bei großen Software-Projekten schon oft zu schwerwiegenden und kostenintensiven Fehlern und Ausfällen geführt hat. (vgl. etwa [honl08])

Durch den Ausfall eines e-Business-Systems frustrierte Benutzer bedeuten für die Betreiber Kundenverlust und damit entgangene Gewinne. Des Weiteren kann gerade in zeitkritischen Anwendungsbereichen wie dem Gesundheitswesen oder im Börsengeschäft die Überlastung eines Systems fatale Folgen haben. Deutlich wird die Relevanz dieses Problems unter Anderem daran, dass Benutzer zu geringe Geschwindigkeit und langsam ladende Werbebanner als die häufigsten Probleme im Internet sehen – und in der Folge lieber bei einem anderen Händler einkaufen, der ihre Performance-Ansprüche erfüllen kann [Corp98]. Die Kundenzufriedenheit könnte hier durch erhöhte Performance deutlich gesteigert werden. Es ist jedoch auch auf Anbieterseite festzustellen: Performance-Probleme sind die zweithäufigste Ursache für das Scheitern eines Software-Projekts [Glas98].

Gerade zu Zeiten außergewöhnlich hoher Belastung muss ein System dem Ansturm an Anfragen gewachsen sein. Es waren in der Vergangenheit bereits mehrere Ausfälle großer Internetanbieter besonders in der hochfrequentierten Vorweihnachtszeit zu bemerken, die große Einnahmeverluste und Kundenfluktuation nach sich zogen. Wenn solch ein Ausfall einen anbieterübergreifenden Service wie eine Kreditkarten-Prüfstelle betrifft, potenzieren sich die wirtschaftlichen Effekte zusätzlich. Unzureichende Planung und mangelnde Skalierbarkeit lassen dieses Phänomen Jahr [SaWP99] für Jahr [Adco07] wieder auftreten.

Dem jüngeren Publikum wird auch noch das Beispiel studiVZ bekannt sein. Das studentische Community-Portal hatte über Monate mit Performance-Problemen und längeren Ausfallzeiten zu kämpfen, da die bereitgehaltenen Server-Ressourcen dem wachsenden Ansturm von Benutzern nicht standhalten konnten. Die darunter liegende Software musste komplett neu geschrieben werden [Klei06]. Eine vorausschauendere Planung und besser durchdachte Skalierbarkeits-Möglichkeiten hätten hier viel Zeit, Geld und Ärger auf Benutzer- wie auf Anbieterseite ersparen können.

1.2 Gliederung

Die Arbeit teilt sich im Folgenden in fünf Kapitel auf.

Nach der einleitenden Motivation in Abschnitt 1.1, wird in den weiteren Ausführungen dieses Kapitels 1 die Voraussetzungen und grundlegenden Begriffe zum Thema erläutert. In Abschnitt 1.3 wird eine kurze Einführung in Dienstgüte und nicht-funktionale Anforderungen gegeben. Warum Entscheidungen bezüglich der Ressourcen-Planung bereits in den frühesten Phasen der Software-Entwicklung berücksichtigt werden sollten, erläutert Abschnitt 1.4.

Anschließend werden in Kapitel 2 die Grundlagen der Anforderungsdefinition und Leistungsgrößen vorgestellt. Grundbegriffe und Gesetze, die zum weiteren Verständnis nötig sind, werden in Abschnitt 2.1 eingeführt. Daraufhin gibt Abschnitt 2.2 einen Überblick über die zu beachtenden Beschreibungsarten eines Systems. Abschnitt 2.3 erläutert die Notwendigkeit der Klassifizierung von Aufträgen, die an das System gestellt werden.

Kapitel 3 zeigt anschließend verschiedene Arten von Modellen der Performance-Berechnung auf. In den Abschnitten 3.1 bis 3.4 werden die Modelle der Markov-Ketten, Warteschlangen-Netzwerke, Prozess-Algebren sowie Petri-Netze eingeführt. Den Beschreibungen der ersten beiden Modelle wird dabei etwas mehr Beachtung geschenkt, da sie die Grundlagen für die darauf folgenden Varianten bilden. In Abschnitt 3.5 wird eine Erweiterung der Unified Modeling Language (UML) vorgestellt, die auch mit dieser Sprache performance-sensitive Modellierung möglich macht.

Zum Schluss soll in Kapitel 4 einen etwas weitergehenden Blick in die Praxis am Beispiel des Vergleichs zweier Hochleistungsrechner erfolgen, bevor mit Kapitel 5 eine Zusammenfassung sowie ein Ausblick auf weitere Entwicklungen gegeben wird.

1.3 Dienstgüte und -vereinbarungen in Software-Projekten

Aus den genannten Beispielen lässt sich ableiten, dass ein Software-Projekt nur dann erfolgreich zum Einsatz gebracht werden kann, wenn neben seiner reinen Funktionalität auch die Performance bezüglich der gestellten Belastbarkeitsbedingungen gewährleistet wird. Eine Definition der Qualitätsmaßstäbe für im Vorfeld definierte Belastungssituationen des Systems ist unumgänglich um einen reibungslosen Ablauf zu garantieren. Die sog. *Dienstgüte* (Quality of Service, QoS) dient dabei als Menge von Qualitätsanforderungen des Benutzers an Leistung, Verfügbarkeit, Sicherheit und Wartbarkeit [MeAD04, S. 11], die durch das System einzuhalten sind. Diese Bedingungen werden als *nicht-funktionale Anforderungen* bezeichnet [CNYM99, S. 1].

Die Zusicherung, diese Qualitätsmaßstäbe auch einzuhalten, stellt die *Dienstgütevereinbarung* (Service Level Agreement, SLA) dar: Eine Vereinbarung wird zwischen Leistungsempfänger und -erbringer getroffen, über eine „definierte [Dienst-] Leistung mit einer bestimmten Qualität (Service Level) in einem bestimmten Umfang“ [Klin05, S. 465]. Da sich nun der Dienstanbieter dazu verpflichtet, die

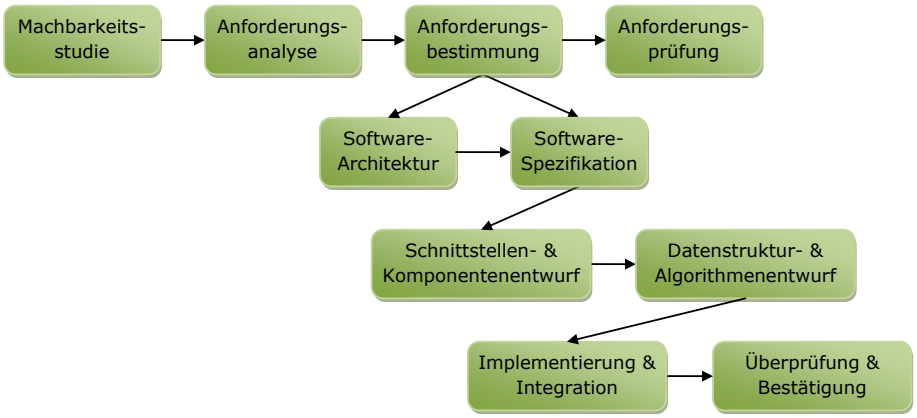


Abbildung 1. Allgemeines Software-Lebenszyklus-Modell [BMIS04, S. 298].

geschuldete Leistung zu erbringen, liegt es in seinem Interesse, dies auch einzuhalten, da er sonst mit Sanktionen zu rechnen hat [Walt06, S. 11]. Durch Vorhalten ausreichender Ressourcen, kann dauerhafte Erreichbarkeit und Performance eines angebotenen Dienstes sichergestellt werden.

Es ist nun notwendig, einen geeigneten Mittelweg zwischen ausreichender Bereitstellung von Ressourcen und ungenutzten Leistungsreserven zu finden. Dieser Prozess wird als *Sizing* bezeichnet.

1.4 Sizing und Performance als Entwurfsentscheidung

Performance- und Skalierbarkeitsziele, die bereits in den ersten Phasen des Lebens-Zyklus eines Systems (vgl. Abbildung 1) einberechnet wurden, sind zu späterem Zeitpunkt deutlich leichter zu erreichen [MeAD04, S. 35].

Eine zu grobe oder fehlerhafte Abschätzung von Anforderungen an ein System kann zu gravierenden Engpässen oder System-Überlastungen bei zu knapp bemessenen Ressourcen führen. Auf der anderen Seite führt eine zu großzügige Dimsionierung zu unnötig hohen Investitionskosten. Die Notwendigkeit, anstatt der vielfach eingesetzten „Pi mal Daumen“-Regeln und dem Eingreifen erst nach Auftreten von Engpässen, bereits frühzeitig wissenschaftlich fundierte Berechnungen über Belastungssituationen und benötigte Ressourcen anzustellen, ist dabei schon lange bekannt, wie etwa [LaCh87, S. 8] beweist:

> Es offenbart sich eine bedeutende Lücke zwischen Theorie und Praxis bezüglich der Anwendung der wissenschaftlichen und ausgereiften Techniken. Diese stellen eine unerlässliche Funktion der Kapazitätsplanung dar, die bei richtigem Einsatz, dem Unternehmen mehrere Millionen Dollar ersparen könnte.

Die Möglichkeit der Erstellung eines Prototyps zur Planung und Abschätzung, wie sie zum Beispiel im Automobilbau gegeben ist, stellt hier zwar eine

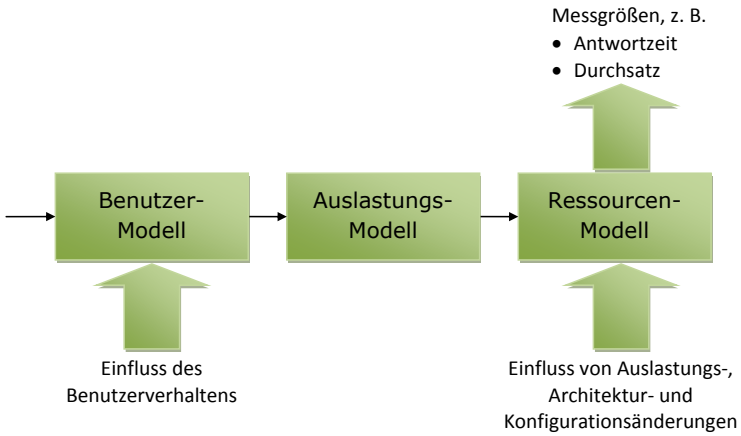


Abbildung 2. Benutzer-, Auslastungs- und Ressourcenmodell. (angelehnt an [MeAl00, S 43])

sehr exakte Alternative dar, steht jedoch aus Komplexitäts-, Zeit- und Kostengründen oft nicht zur Verfügung. Es ist ein ingenieurmäßiges Herangehen an die Problemstellung notwendig, die mit der Statik-Berechnung im Brückenbau verglichen werden kann: Werden Stabilität, Gewicht und Kraftverteilung dort nicht exakt genug von vornherein berechnet, kann das gesamte System auf Dauer nicht funktionieren und unter Überlast einbrechen.

Abbildung 2 zeigt die Einordnung des Ressourcen-Modells in das Gesamtkonzept der Einflussmodelle auf ein Software-System. Es ist die Beziehung der verschiedenen Einflüsse durch Benutzerverhalten auf der einen, und die Einflüsse durch Belastungssituationen, Architektur und Konfiguration auf der anderen Seite, dargestellt. Vorhersagen über Häufigkeit und Komplexität der an das System gestellten Anfragen können mit dem Wissen über das Benutzerverhalten demnach sehr viel besser eingeschätzt werden.

Es ist also ratsam, so früh wie möglich im Entwicklungszyklus eines Software-Projektes an Performance-Entscheidungen zu denken. Weiterhin ist zu beachten, dass diese in den weiteren Stufen immer wieder überprüft, aktualisiert und angepasst werden müssen. So können grobe Abschätzungen durch nun genauer bekannte Größen ersetzt und so die Zuverlässigkeit der Vorhersagen gesteigert werden [NKPP+00, S. 230].

Bei der Modellierung einer Anforderungssituation ist eine Konzentration auf das Wesentliche notwendig: Komplexe Systeme werden so vereinfacht, dass alle relevanten Größen einen Platz im Modell finden, unwichtige Faktoren jedoch schematisch abgekürzt oder unterdrückt werden [BoRi89, S. 1 f.].

Einen Überblick über die verschiedenen Methoden der Leistungsanalyse gibt Abbildung 3. Im Weiteren wird hierbei auf den Bereich der Modellierungstechniken eingegangen.

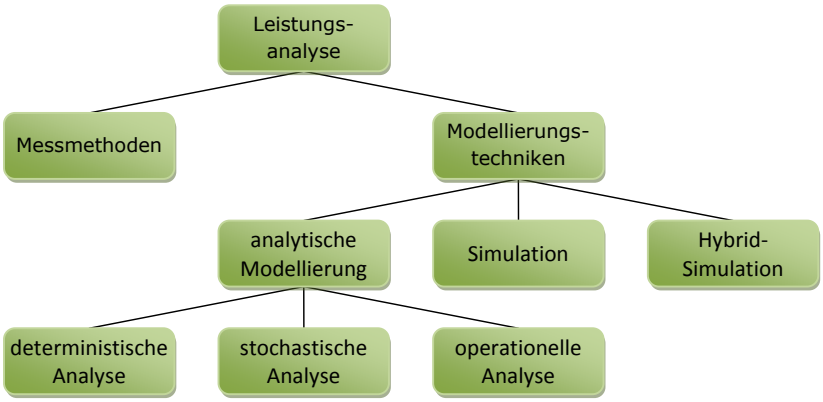


Abbildung 3. Methoden der Leistungsanalyse [HaZo95, S. 16].

Die Planungen bezüglich der bereitgestellten Kapazitäten können sowohl für ein neu zu erstellendes System, als auch für die Erweiterung eines bestehenden Systems um neue Anwendungen und Funktionen erfolgen. Abbildung 4 stellt für den Fall der Erweiterung den Zusammenhang zwischen bestehender Belastung durch bereits vorhandene Aufträge und den hinzukommenden neuen Belastungen an das System dar. Erstere können dabei durch Messung mit Hard- und Software-Monitoren quantifiziert werden, letztere müssen durch die vorgestellten Methoden modelliert und anschließend bewertet werden.

2 Grundlagen der Anforderungsdefinition und Leistungsgrößen

2.1 Relevante Grundbegriffe und Gesetze

Das Ziel der Modellierung eines Systems ist nun die Bestimmung von Leistungsgrößen, die die Performance des Systems beschreiben. In Anlehnung an [Beil81, S. 2], soll *Performance* durch die folgende Gleichung beschrieben werden:

$$(Hardware + Anwendung) \times Arbeitslast \longrightarrow System\text{-}Performance \quad (1)$$

Es lässt sich also aus der Kombination eines bestimmten Systems, bestehend aus Hardware-Ressourcen und der Software-Anwendung, mit einer bestimmten Arbeitslast ein bestimmtes Maß an Performance ableiten. Die dabei relevanten Größen werden im Folgenden erläutert.

Bei der Bestimmung aller Leistungsgrößen wird in dieser Arbeit von einem stabilen System ausgegangen, das sich im *statistischen Gleichgewicht* befindet. Eventuelle Einschwingvorgänge, die beim Start des Systems auftreten, sind also bereits abgeklungen. Die Rate der eingehenden Aufträge muss somit der Rate der ausgehenden Aufträge entsprechen. Ist dies nicht der Fall und es kommen

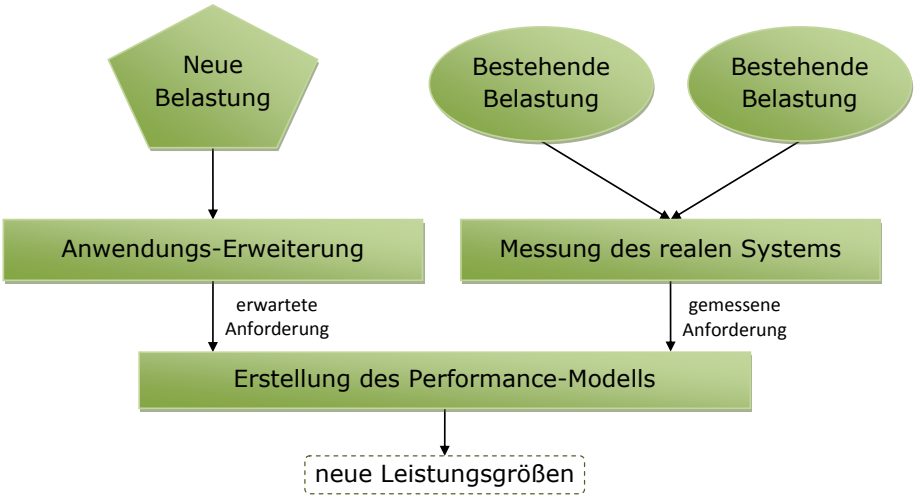


Abbildung 4. Kapazitätsplanung bei System-Erweiterungen. (angelehnt an [MeAD94, S. 342])

mehr Aufträge pro Zeiteinheit an, als bearbeitet werden können, läuft das System voll und eine sinnvolle Berechnung ist nicht mehr möglich [BoRi89, S. 40]. Der Zustand außerhalb des Gleichgewichts soll hier nicht betrachtet werden.

Um im Weiteren über die wichtigsten Grundbegriffe der Performance-Berechnung im Klaren zu sein, folgt eine kurze Sammlung der relevanten Formeln und Gesetze mit Erläuterungen, wie sie in der Literatur üblicherweise zu finden sind. Sie beziehen sich dabei auf die in Abschnitt 3.2 weiter erläuterten Warteschlangen. (vgl. [MeAD04, S. 13 ff., 37], [LaCh87, S. 25 ff.], sowie die deutschen Begriffe in [BoRi89, S. 40 f.])

Bedienzeit
Zeit, die die reine Bearbeitung eines Auftrags benötigt.

$$Bedienzeit = \frac{Auslastung}{Durchsatz} \tag{2}$$

Wartezeit
Zeit, die ein Auftrag warten muss, bevor seine Bearbeitung beginnt.

Antwortzeit
Zeit zwischen Eingang und vollständiger Bearbeitung des Auftrags.

$$Antwortzeit = Wartezeit + Bedienzeit \tag{3}$$

Ankunftsrate
Durchschnittliche Anzahl von Aufträgen, die pro Zeiteinheit im System eintreffen.

Durchsatz/Bedienrate/Abgangsrate
Durchschnittliche Anzahl von Aufträgen, die pro Zeiteinheit im System bedient werden und es somit anschließend verlassen.

Auslastung
Anteil an der Gesamtzeit, während der die Bedieneinheit aktiv (belegt) ist.

$$Auslastung = \frac{Ankunftsrate}{Bedienrate} \tag{4}$$

Eine alternative Berechnungsmethode bietet das Auslastungsgesetz.

$$Auslastung = Ankunftsrate \times mittlere\ Bedienzeit \tag{5}$$

Noch hinzuzufügen ist das *Gesetz von Little* [Litt61], mit dem leicht berechnet werden kann, wie viele Aufträge sich aktuell im System befinden,

$$mittlere\ Anzahl\ der\ Aufträge\ im\ System = Durchsatz \times mittlere\ Bedienzeit \tag{6}$$

und das Gesetz der Service-Forderung, das die Summe aller Bedienzeiten errechnet.

$$Service\text{-}Forderung = \frac{Auslastung}{Durchsatz} \tag{7}$$

Es ist jeweils zu beachten, wie aussagekräftig die genannten Durchschnittswerte sind. Je nach Anwendungsfall kann die Art der Verteilungsfunktion von Ankunfts- und Bedienrate variieren.

Die Beschreibung dieser Verteilungen erfolgt in der Regel nach der *Kendall-Notation*. Diese ist wie folgt strukturiert:

$$A/B/m$$

Dabei steht A für die Verteilung der Ankunftszeiten, B beschreibt die Verteilung der Bedienzeiten und m gibt die Anzahl identischer, paralleler Bedieneinheiten an, die zur Berechnung zur Verfügung stehen. Mögliche Verteilungsfunktionen erhalten einen repräsentativen Kennbuchstaben zur Identifizierung. Die am häufigsten Verwendeten sind etwa Exponentialverteilung (M), Erlang-Verteilung (E_k) und die deterministische Verteilung (D) mit konstanten Ankunfts- und Bedienzeiten [BoRi89, S. 38]. So hätte beispielsweise ein $M/M/1$-System exponentialverteilte Ankunfts- und Bedienzeiten und eine einzelne Bedieneinheit.

Genauere und weitergehende Ausführungen hierzu gibt z. B. [BoRi89, S. 29 ff., 38 ff.] und [HaZo95, S. 40 f.]. Dort werden differenzierte statistische Analysen aufgeführt und Zufallsvariablen mit deren Verteilungsfunktionen ausführlich erläutert.

2.2 Beschreibung eines System-Modells

Bei der Modellbeschreibung und -erstellung gilt es, das zu bewertende System in einer abstrakten Darstellung zu formalisieren und mathematisch handhabbar zu machen. Wie bereits aufgezeigt, ist darauf zu achten, sich auf diejenigen

Eigenschaften eines Modells zu beschränken, die für die jeweilige Fragestellung von Bedeutung sind (vgl. Abschnitt 1.4). Die drei Beschreibungsarten sind im Folgenden aus [BoRi89, S. 7 ff.] entnommen.

Bei der *Konfigurationsbeschreibung* wird das Rechner-System mit seinen wichtigsten Komponenten quantitativ beschrieben und deren Interaktionen erfasst. Hierzu zählen etwa:

System-Komponenten
: Art und Anzahl von Prozessoren mit ihren Leistungsgrößen, Haupt- und Massenspeicher, sowie deren Kapazitäten.

System-Struktur
: Verbindungswege wie z. B. Busse.

Bearbeitungs- und Verwaltungsstrategien
: Speicherverwaltungsstrategien, Scheduling, ...

Die *Lastbeschreibung* charakterisiert das dynamische Ablaufgeschehen in einem Rechner-System. Das zu erwartende Benutzerverhalten (Ankunftsprozess) und die resultierende typische Auftragsstruktur (Bedienprozess) ist zu identifizieren und zu beschreiben. Hierbei ist es meist dienlich, die Auftragslasten in Auslastungsklassen einzuteilen, um so einen Mittelweg zwischen Standardisierung und Individualität von Benutzeranfragen in einem handhabbaren Modell zu finden. Auf diese Klassifizierung wird in Abschnitt 2.3 genauer eingegangen. Hierbei stellt jedoch gerade das Benutzerverhalten eine schwer vorhersagbare Größe dar. Eine sog. „interaktive Last“, bei der die direkte Interaktion mit dem Benutzer und seine Denkzeit großen Einfluss auf das System-Verhalten hat, stellt so eine schwierige Modellierungssituation dar.

Nach Erfassung der vorgenannten Konfigurations- und Lastbeschreibung, können diese Größen mit der *Modellbeschreibung* in Zusammenhang gebracht werden. Sie bildet mit der *Warteschlangentheorie* eine „wahrscheinlichkeitstheoretische Beschreibungstechnik“ ([BoRi89, S. 9]), die auf den folgenden Grundsätzen basiert.

- Komplexe, deterministische Vorgänge können durch *stochastische Regularitäten* erfasst werden.
- *Grundlegende Unbestimmtheiten* sind für die zu beschreibenden Abläufe charakteristisch.
- Es stehen zum aktuellen Zeitpunkt *nicht genügend Informationen* über die Bedingungen der Abläufe zur Verfügung.

2.3 Einteilung von Auslastungsklassen

Anwendungsgebiete von IT-Systemen sind heutzutage sehr verschiedenartig. Es lässt sich jedoch feststellen, dass bei fast all diesen Systemen verschiedene Klassen von Auslastungstypen zu finden sind, die sich hinsichtlich ihrer relativen Häufigkeit und ihrer Komplexität, und damit in ihrem Rechenaufwand, unterscheiden. So wird z. B. eine kurze Datenanfrage an einen Datenbankserver weit

häufiger auftreten als eine komplexe Datensicherung. Des Weiteren können Aufträge hinsichtlich ihres Typs (z. B. Lese- und Schreibzugriffe) und ihrer geforderten Dienstgüte (vgl. Abschnitt 1.3) klassifiziert werden.

Eine Einteilung in Klassen ist oft sinnvoll, um genauere Vorhersagen über das System-Verhalten machen zu können. Ohne diese Klassifizierung wäre das entstehende Modell zu grob und würde nur unzureichende Ergebnisse liefern [MeAD04, S. 41]. Die optimale Granularität der Klassen variiert dabei von System zu System.

Die Charakterisierung von Auslastungsklassen lässt sich nach [Koun05, S. 132 ff.] in den folgenden fünf Schritten durchführen:

1. Identifizierung der Grundbestandteile der Auslastung.
2. Aufteilung von Grundbestandteilen ähnlichen Charakters in Auslastungsklassen.
3. Identifizierung der System-Komponenten und -Ressourcen, die von den Auslastungsklassen genutzt werden.
4. Beschreibung von Wechselbeziehungen zwischen den einzelnen Komponenten und von Verarbeitungsschritten für jede Auslastungsklasse.
5. Charakterisierung der Auslastungsklassen bezüglich ihrer Dienstanforderungen und Auslastungsintensität.

Tabelle 1 gibt ein Beispiel für eine dreiklassige Einteilung für einfache, mittlere und komplexe Aufträge an einen Datenbank-Server mit ihrer jeweiligen relativen Häufigkeit sowie den Belastungsanforderungen an Prozessor (benötigte Bedienzeit) und Festplatte (Anzahl von Zugriffen).

Auslastungsklasse	relative Häufigkeit	∅ Bedienzeit CPU (Sek.)	∅ Anz. Zugriffe
Einfach	45%	0,04	5,5
Mittel	24%	0,18	28,9
Komplex	30%	1,20	85,0

Tabelle 1. Auslastungsklassen eines Datenbank-Servers [MeAD04, S. 41].

Die oben genannten Schritte können hierbei wie folgt identifiziert werden:

1. Es handelt sich bei den genannten Klassen um Datenbankanfragen.
2. Es wurden drei Klassen nach ihrem Ressourcenverbrauch identifiziert.
3. Bei den genutzten Ressourcen sind CPU und Festplatte zu nennen.
4. Ein Diagramm für die Ressourcen-Nutzung wird später mit Abbildung 6(a) vorgestellt.
5. Die Intensität der Auslastung ist in den Spalten „∅ CPU-Zeit (Sek.)“ und „∅ Anz. I/Os“ eingetragen.

3 Arten der System-Modellierung

3.1 Markov-Ketten

Eine weit verbreitete Methode, um Performance-Modelle darzustellen, sind *Markov-Ketten*. Jeder Zustand in der Kette repräsentiert hier einen möglichen Zustand, in dem sich das zu untersuchende System befinden kann. Von ihm aus kann mit stochastisch verteilten Übergangswahrscheinlichkeiten ein anderer Zustand folgen (Transition).

Als Beispiel soll hier ein Server mit einem Prozessor (*CPU*) und zwei unterschiedlich schnellen Festplatten (langsamere *Festplatte 1*, schnellere *Festplatte 2*) dienen (s. Abbildung 5(a)). Es befinden sich im Untersuchungszeitraum zwei Benutzer im System, die jeweils eine Komponente nutzen. Die zugehörige Markov-Kette ist in Abbildung 5(b) als endlicher Automat dargestellt. Die Übergangswahrscheinlichkeiten an den Transitionen stellen hierbei Flussraten dar, die durch Leistungskennzahlen der Komponenten vorgegeben sind.

Der exemplarische Prozessor kann etwa 6 Anfragen pro Minute bearbeiten (Bedienrate), die sich zu gleichen Teilen auf die Festplatten aufteilen – die jeweiligen Flussraten an den Übergängen zu den Festplatten ergeben sich zu 3.

Dieses Minimalbeispiel gibt einen Einblick, wie Systeme mit Markov-Ketten modelliert werden können. Zur Lösung gilt es nun noch, Gleichungen der Zustandswahrscheinlichkeiten P_i aufzustellen, die ein Gleichgewicht im System beschreiben.

P_i: Wahrscheinlichkeit, dass sich das System in Zustand i befindet.

Jeder Zustand erhält eine solche Gleichung, so etwa Gleichung 8 für den Zustand $(2, 0, 0)$, an dem sich beide Benutzer an der CPU befinden.

$$(4 \times P_{(1,1,0)}) + (2 \times P_{(1,0,1)}) = (6 \times P_{(2,0,0)}) \tag{8}$$

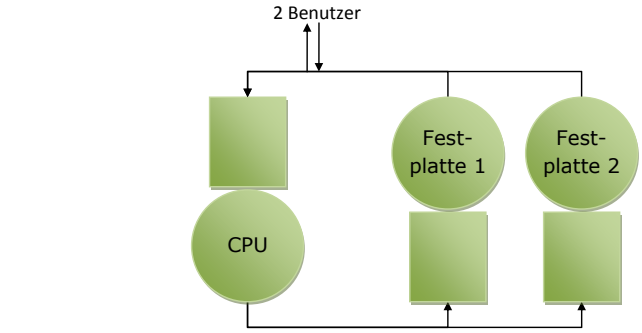
n Zustände ergeben so n Gleichungen mit n Variablen. Nach Auflösen der Gleichungen, was einen trivialen mathematischen Schritt darstellt, erhält man für jeden Zustand eine feste Zustandswahrscheinlichkeit P_i.

$$P_{(2,0,0)} = \frac{16}{115} \approx 0,1391 \tag{9}$$

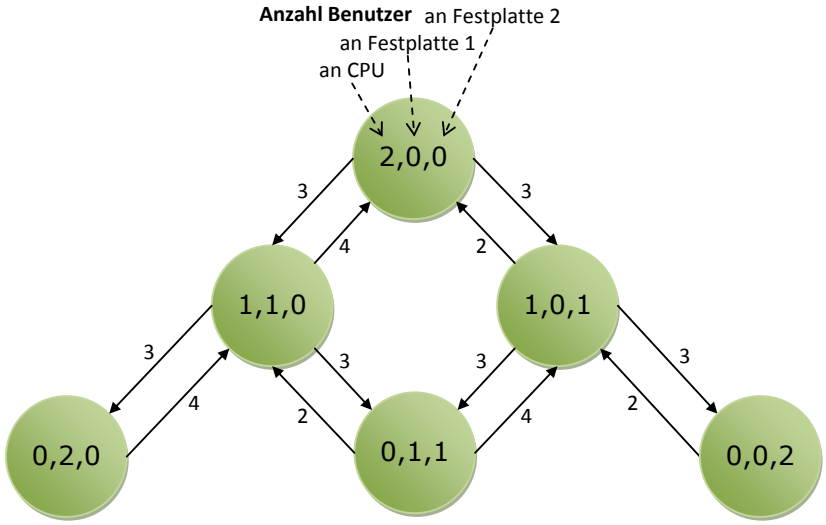
Aus diesen Zustandswahrscheinlichkeiten können nun eine Vielzahl von Leistungsgrößen des Gesamt-Systems nach den in Abschnitt 2.1 vorgestellten Gesetzen abgeleitet werden. Aussagen über Antwortzeiten, Komponentenauslastung, etc. des modellierten, aber auch eines modifizierten Systems lassen sich hier leicht ablesen [MeAD04, S. 274 ff.].

Beispielhaft sei die CPU-Auslastung aus obigem Beispiel genannt. Diese berechnet sich über die Summe der Zustandswahrscheinlichkeiten, an denen der Prozessor aktiv ist.

$$Auslastung_{CPU} = P_{2,0,0} + P_{1,1,0} + P_{1,0,1} \approx 0,4521 \tag{10}$$



(a)



(b)

Abbildung 5.
(a) Datenbank-Server mit zwei Festplatten.
(b) Zugehörige Markov-Kette. (angelehnt an [MeAD04, S. 261, 268])

Wird diese nun mit der Bedienrate der CPU multipliziert, erhält man den Durchsatz.

$$Durchsatz_{CPU} = 0,4521 \times 6 = 2,7126 \tag{11}$$

Bei zwei Benutzern im System ergibt sich nach dem Gesetz der Service-Forderung (vgl. Gleichung 7):

$$Service\text{-}Forderung_{CPU} = \frac{2}{2,7126} \approx 0,7373\,min \approx 44,24\,sec \tag{12}$$

Wie zu sehen ist, lassen sich wichtige Performance-Indikatoren sehr schnell aus den aufgestellten Gleichungen ableiten. Ein Problem hierbei ist, dass die Anzahl der Zustände, und damit die der Gleichungen, mit der im System befindlichen Benutzerzahl exponentiell wächst – was sehr schnell zu komplexen Darstellungen führt. Die Anzahl an Zuständen liegt bei $\binom{N+K-1}{K-1}$, was für ein kleines System von 25 Benutzern an 25 Computern bereits zu über 63 Milliarden, bei je 50 zu $5 \cdot 10^{28}$ Zuständen führt.

Abhilfe schafft hier die Technik der *„mean value analysis“* (MVA, Mittelwert-Analyse), die durch die Annahme von Durchschnittswerten und rekursives Vorgehen die Berechnung für jede beliebige Benutzerzahl möglich macht [Kozi04, S. 10 f.]. Dabei wird jeweils von Mittelwerten bei $n-1$ Benutzern im System ausgegangen und iterativ auf die Werte für n Benutzer geschlossen. Dieser Schritt hat einen marginalen Rechenaufwand und ist so auch für umfangreiche Markov-Ketten anwendbar [ReLa80, S. 314].

Für detailliertere Betrachtungen des MVA-Algorithmus sei auf [ReLa80] und die Beschreibungen in [MeAD04, S. 311 ff.] verwiesen.

3.2 Warteschlangen-Netzwerke

Warteschlangen-Netzwerke stellen eine auf Markov-Ketten basierende Modellierung dar. Ihr liegt zu Grunde, dass auf eine einzelne *Bedienstation* mehrere Anfragen treffen und sich diese in einer Warteschlange aufreihen, bis sie an einer *Bedieneinheit* bearbeitet werden können.

Die Theorie hinter Warteschlangen-Netzwerken ist schon weit älter als die Welt der Computer. [Jack57] beschrieb sie erstmals 1957, ihre Anwendung auf Computer erfolgte dann ein paar Jahre später [Sche65]. Sie hat jedoch bis heute große Bedeutung in der Untersuchung von Systemen und Computermodellen.

Warteschlangen-Netzwerke bestehen in ihrer einfachsten Form aus *Warteschlangen* und *Bedieneinheiten*. Als Beispiel soll hier wiederum ein Datenbank-Server dienen. Das Modell wurde, wie in Abschnitt 1.4 gefordert, auf die für die Performance wichtigen Bausteine beschränkt. Anfragen werden an den Server geschickt, vom Prozessor ausgewertet, Zugriffe auf die Festplatte getätigt und abgearbeitete Anfragen aus dem System entlassen. Abbildung 6(a) gibt eine grafische Übersicht über die Vorgänge in diesem System.

Bei der Abarbeitung der empfangenen Aufträge kommt eine *Warteschlangendisziplin* (auch *Bedienstrategie*) zum Einsatz, um die Bearbeitungsreihenfolge zu

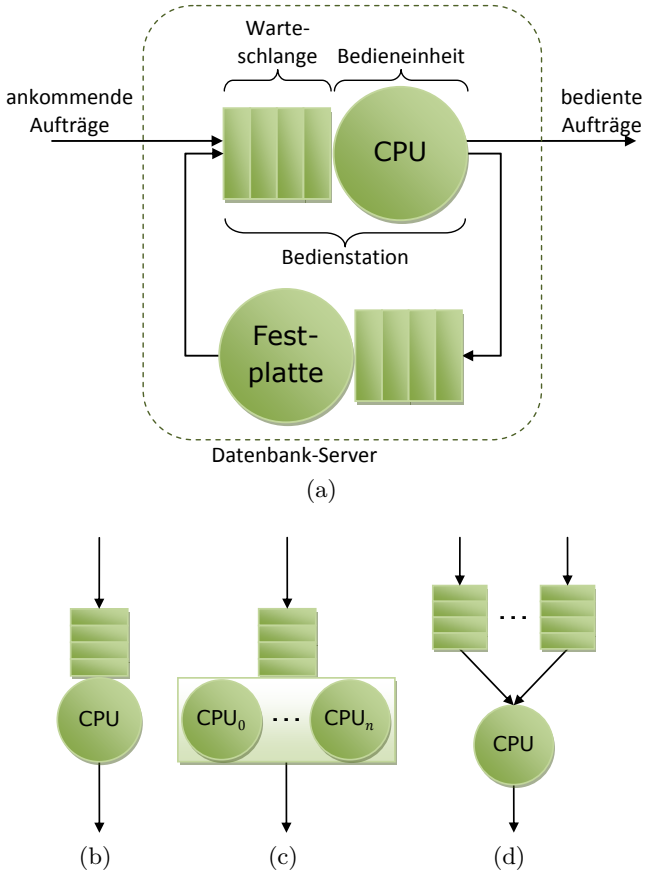


Abbildung 6.
(a) Warteschlangenmodell eines Rechner-Systems.
(b) Einfaches Warte-System.
(c) Warte-System mit gemeinsamer Warteschlange.
(d) Warte-System mit gemeinsamer Bedieneinheit.
(angelehnt an [MeAD04, S. 39 f.])

bestimmen. So können beispielsweise diejenigen Aufträge zuerst bearbeitet werden, die als Erstes an der Warteschlange eintreffen („first come first serve“) oder aber es wird immer ein kleiner Teil eines jeden Auftrags abgearbeitet („round robin“). Diese und weitere Strategien werden etwa in [BoRi89, S. 38 f.] beschrieben. Welche dabei die passende Wahl darstellt, hängt immer von dem zu modellierenden System ab und sollte bei jeder Komponente analysiert werden.

Hat eine Bedienstation mehrere Bedieneinheiten, jedoch nur eine Warteschlange, sollte dies natürlich im Modell nachgebildet werden. Eine Multikern-CPU, wie in Abbildung 6(c) verdeutlicht, ist hier ein einleuchtendes Beispiel, das auch im Datenbank-Server von Abbildung 6(a) zur Anwendung kommen könnte. Auch der umgekehrte Fall kann zutreffend sein: Mehrere Warteschlangen benutzen eine gemeinsame Bedieneinheit. Dies kann genutzt werden, um verschiedene Prioritätsklassen zu modellieren.

Die Modellierung von Warteschlangen kann nun *analytisch* oder durch *Simulation* erfolgen.

Das analytische Herangehen bedeutet hier, dass Beziehungen zwischen System- und Leistungsgrößen mathematisch hergeleitet werden. Dabei wird unterschieden zwischen *stochastischen* Modellen, die statistisch verteilte Parameter (Ankunftsrate, Bearbeitungszeit) annehmen, und *operationellen* Modellen, die Durchschnittswerte und beobachtete Größen als Grundlage heranziehen.

Simulationen dagegen nutzen Computerprogramme um System-Struktur und -verhalten zu untersuchen. Eingabegrößen bilden hier vorher festgelegte oder zufällig bestimmte Belastungssituationen.

Die durchschnittliche Antwortzeit berechnet sich dabei für ein Simulationsmodell mit n Aufträgen über die einzelnen Antwortzeiten.

$$Antwortzeit = \frac{\sum_{i=1}^{n} Antwortzeit_i}{n} \tag{13}$$

Bei analytischen Modellen lautet die Berechnungsformel entsprechend

$$Antwortzeit = \frac{Bedienzeit}{1 - (Ankunftsrate \times Bedienzeit)} \tag{14}$$

Beide Alternativen zeigen in ihrer Anwendung jeweils gegensätzliche Vor- und Nachteile, die je nach Einsatzzweck gegeneinander abzuwägen sind: Einen wichtigen Punkt stellt der Aufwand dar, der zur Erstellung des Modells notwendig ist. Analytische Modelle sind in der Regel kostengünstiger und weniger zeitaufwändig, dabei effizienter in der Berechnung und, durch ihren höheren Abstraktionsgrad, leichter mit Eingabeparametern zu versehen. Simulationen hingegen stellen in anderen Bereichen die bessere Alternative dar, da sie gerade so detailliert gestaltet werden können, wie es für die Anwendungssituation nötig ist. Hierbei können realistischere Annahmen getroffen werden. Des Weiteren sind manche System-Details nur durch Simulation darzustellen, da analytische Modelle hier fehlen oder bei zu hoher Komplexität unmöglich sind [BoRi89, S. 2 f.][MeAD04, S. 36 f.].

In Mischformen, sog. Hybrid-Simulationen, wird versucht, die Vorteile der Methoden in verschiedenen Teil-Systemen auszunutzen und im Gesamt-System

	Hybrid	SIM1	SIM2	SIM3
Test 1	0,365	30,567	30,832	31,022
Test 2	0,554	42,141	42,313	42,390
Test 3	0,664	45,546	45,219	46,252
Test 4	2,634	46,886	47,191	46,620
Test 5	0,270	53,714	53,331	53,425

Tabelle 2. Vergleich der Rechenzeit (in Sek.) für die Ausführung der Modelle bei gewöhnlicher Simulation (*SIM1*, *SIM2*, *SIM3*) und Hybrid-Simulation (*Hybrid*) [Schw78, S. 721].

	Hybrid	SIM1	SIM2	SIM3
	(Sek.)	(%)		
Test 1	1260,1	-1,1	-0,1	0,9
Test 2	639,6	-3,2	-0,7	2,2
Test 3	663,9	-1,5	-1,1	2,9
Test 4	15,5	-8,4	-12,9	4,5
Test 5	1472,4	-1,1	-0,6	-2,9

Tabelle 3. Errechnete Leistungsgröße (mittlere Antwortzeit in Sek.) bei Hybrid-Simulation (*Hybrid*) und Abweichung (in %) bei gewöhnlicher Simulation (*SIM1*, *SIM2*, *SIM3*) [Schw78, S. 721].

positiv zu addieren. Für die Berechnung verschiedener Beispielmodelle beschreibt [Schw78, S. 721 f.] eine Zeiteinsparung um den Faktor 100 für die Hybrid-Variante gegenüber reinen Simulationsmodellen (vgl. Tabelle 2).

Die erlangten Ergebnisse (Wartezeit, Bedienzeit, Auslastung usw.) haben dabei nur geringe Abweichungen (je nach Lastsituation zwischen 0,6 und 12,9%, vgl. Tabelle 3) und liegen in einem für die meisten Projekte vertretbaren Rahmen.

3.3 Prozess-Algebren

Die Modellierungsform der *stochastischen Prozessalgebren* basiert ebenfalls auf den oben genannten Markov-Ketten. Sie sehen ein System als „Menge von Prozessen [...], die atomare Aktionen ausführen“[Kozi04, S. 15], welche durch Operatoren zusammengesetzt werden. Hierdurch ist es möglich, nebenläufige Systeme und Bisimulationen (d. h. Relationen zwischen sich gleich verhaltenden Zuständen) darzustellen.

Prozess-Algebren vereinen funktionale wie nicht-funktionale Eigenschaften im selben Modell [BMIS04, S. 301].

Für verschiedene Unterformen der Prozess-Algebren ist bereits eine fortgeschrittene Automatisierung der Performance-Berechnung durch entsprechende Tools möglich. Zu nennen sind hierbei „TIPPtool“ [HHKM$^+$00] für die Umgebung ***TI**me **P**rocesses and **P**erformability evaluation* (TIPP), „PEPA Workbench“ [GiHi94] für die ***P**erformance **E**valuation **P**rocess **A**lgebra* (PEPA) und

„The Two Towers“ [BCSS98] für die ***E**xtended **M**arkovian **P**rocess **A**lgebra* (EMPA).

Die Werkzeuge nehmen dabei nach [HeHK02, S. 51] alle exponentialverteilte Zufallsvariablen an, unterscheiden sich jedoch in ihrem Funktionsumfang bezüglich Nichtdeterminiertheit und Prioritätseinteilung. In besagter Quelle befindet sich zudem ein ausführlicher Überblick über Vor- und Nachteile der drei Werkzeuge.

3.4 Petri-Netze

Das Modell der Petri-Netze ist ebenfalls schon seit vielen Jahren bekannt und vielfach angewendet. Bei der Performance-Berechnung werden *stochastische Petri-Netze* eingesetzt, die exponentialverteilte Übergangswahrscheinlichkeiten besitzen.

Wie [Moll82, S. 914] beschreibt, war es bereits vor über 25 Jahren möglich, stochastische Petri-Netze mit 50 Stellen mit Hilfe eines FORTRAN-Programms leicht zu lösen. Verglichen mit den nicht berechenbaren Werten aus Abschnitt 3.1 stellt dies eine erhebliche Vereinfachung dar.

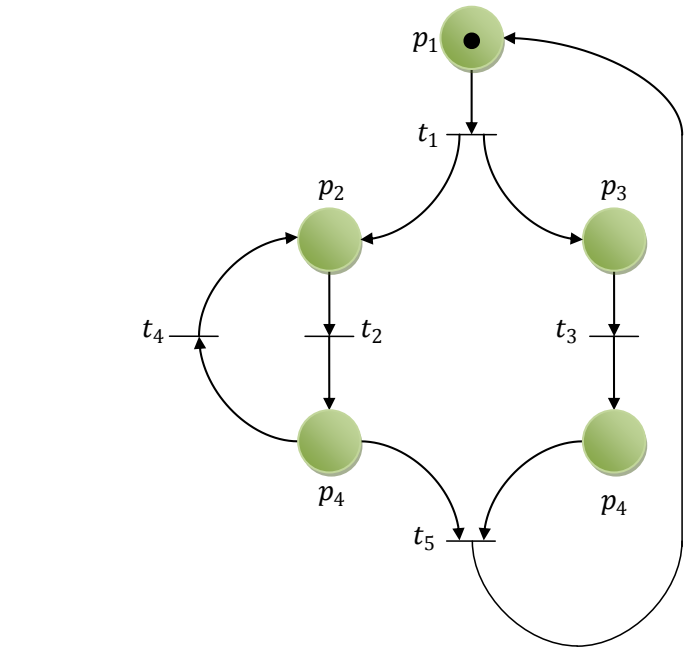
Des Weiteren wird ein Beweis geliefert, dass es für jedes Petri-Netz eine gleichbedeutende (isomorphe) Markov-Kette gibt. Jede Markierungskombination hat hierbei einen entsprechenden Zustand in der Markov-Kette. Ein Beispiel gibt Abbildung 7. Hätte das Petri-Netz hier in der Startmarkierung zwei Marken, so hätte die zugehörige Markov-Kette bereits 14 Zustände, bei drei Marken schon 30 [Moll82, S. 915].

Auch Petri-Netze werden zunehmend automatisiert. Eine Liste mit verfügbaren Frameworks und Tools findet sich z. B. in [HeMo08].

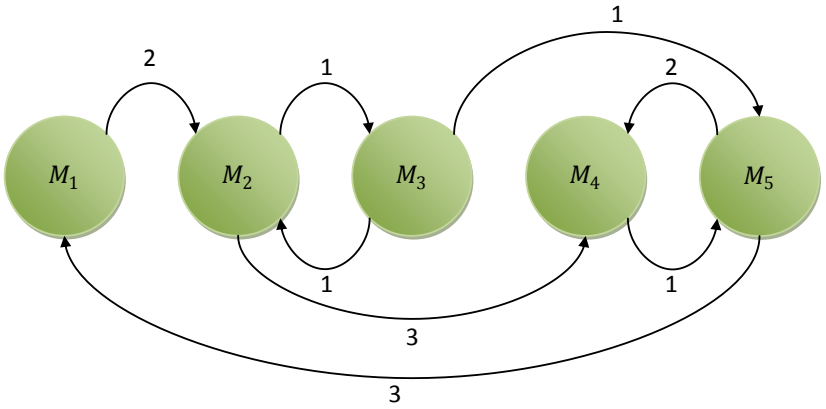
3.5 Unified Modeling Language (UML)

Aus der Softwaretechnik sind verschiedene Methoden bekannt, eine Software vor ihrer Implementierung zu beschreiben. Eine der bedeutendsten dieser Methoden stellt heute die *Unified Modeling Language* (UML) dar. Ursprünglich gehörte es nicht zu ihren Zielen, performance-relevante Begebenheiten zu modellieren, sondern Aufbau und analytischen Ablauf der Systeme darzustellen. Dieses Problem wurde erkannt und mit dem „UML Profile for Schedulability, Performance, and Time Specification“ [Grou03] behoben. (Das Dokument liegt mittlerweile in der aktualisierten Version 1.1 vor [Grou05].) Hierdurch wurde die Modellierung von Echtzeit-Systemen und die Verwendung von Performance-Indikatoren möglich.

Die verwendeten Spezifikationen orientieren sich an den UML-Diagrammtypen Sequenz-, Aktivitäts-, und Zustandsdiagramm. Diese werden zusätzlich um die Möglichkeit erweitert, durch Stereotypen und „constraints“ Performance- und QoS-Faktoren einzubinden [Grou05, S. 2 ff.].



(a)



(b)

Abbildung 7. (a) Petri-Netz mit fünf Stellen, fünf Transitionen und einer Marke. (b) Zugehörige Markov-Kette. ([Moll82, S. 914])

3.6 Automatisierung der Ressourcen-Dimensionierung

Eine automatisiertes Vorgehen, unterstützt durch Software-Werkzeuge, erleichtert die Projektplanung und macht Ressourcen-Dimensionierung für eine große Zahl von Projekten möglich. Die Werkzeuge stützen sich dabei auf eine Datenbank mit einer Vielzahl von verfügbaren Hard- und Software-Systemen mit deren Leistungsgrößen. Nach Eingabe von Kennzahlen der Arbeitslast und der gestellten Anforderungen, berechnet das Werkzeug die notwendigen Ressourcen und vergleicht diese mit Kandidaten aus der Datenbank.

Als Ergebnis wird eine Empfehlung abgegeben, welches System am Besten für die gestellte Arbeitslast geeignet ist. [WaMR04, S. 137].

4 Performance-Vergleich zweier Systeme

Eine etwas andere Anwendung der vorgestellten Methoden kann in [KeHo03] gefunden werden. Um ein Systeme bezüglich seiner Performance zu bewerten bedarf es, wie aus den bisherigen Ausführungen hervorgegangen ist, weit mehr als die bloßen Leistungsgrößen der Einzelkomponenten – jedoch nicht das Vorhandensein bzw. die Verfügbarkeit des Gesamt-System.

Notwendig sind etwa die gestellte Anforderungssituation und, wie gut sie die verfügbaren Ressourcen ausnutzen kann. Diese Bewertung kann dann zum Vergleich zweier verschiedener Systeme herangezogen werden.

Als Beispiel aus der Praxis für einen solchen Vergleich wurden 2003 die damaligen Spitzenreiter der sog. TOP500-Liste der leistungsstärksten Rechner-Systeme der Welt untersucht. Der 40-Teraflop-Höchstleistungsrechner „Earth Simulator“ von NEC wurde mit dem auf Platz zwei gewerteten „ASCI Q“ von HP mit rund 20 Teraflops verglichen. (In der aktualisierten Liste von November 2007 lagen die beiden Supercomputer auf den Plätzen 30 bzw. 91.)

Hierbei wurden die Rechner, anders als für die TOP500-Platzierung nicht mit der Lösung von linearen Gleichungssystemen (LINPACK), sondern den Programmen SAGE sowie Sweep3D getestet. Diese simulieren hydrodynamische Verläufe in Räumen bzw. Neutronentransporte – also hochkomplexe Berechnungen die extrem rechenintensiv, wenn auch sehr verschiedenartig sind.

Das für diese Arbeit Interessante hierbei ist, dass die Programme nicht direkt auf dem Earth Simulator bearbeitet werden konnten, da für diesen Rechner nur ein stark beschränkter Zugang besteht. Es liegen jedoch ausführliche Informationen über den Aufbau der beiden Rechner-Systeme sowie die Struktur der untersuchten Programme vor, sodass es möglich war, die jeweiligen Performance-Indikatoren zu bestimmen. (Im Beispiel waren dies die Verarbeitungsgeschwindigkeit von zu berechnenden Zellzyklen).

Es war festzustellen, dass mit steigender Anzahl von Prozessoren die Durchsatzrate pro Prozessor stetig abnahm. Dies ergab, dass der Earth Simulator, der nur rund ein Drittel der Prozessor-Anzahl besitzt (rund 8000 gegen 5000) bei den Testläufen mit SAGE einen deutlichen Performance-Vorteil aufweisen kann, der auch nach Bereinigung von Vorteilen durch schnellere Einzel-CPU-Leistungen (8 Gigaflops gegen 2,5 Gigaflops) erhalten bleibt. Für den Earth Simulator

wurde hierbei angenommen, dass die Prozessoren über mehr als 10% der Zeit eine maximale Auslastung haben – gegenüber 14% für den ASCI Q.

Bei den Betrachtungen von Sweep3D hingegen kann der Earth Simulator keinen über die höhere FLOP-Rate hinausgehende Performance-Gewinn verzeichnen. Dies wird in [KeHo03] zurückgeführt auf einen schnell wachsenden Kommunikationsaufwand bei steigender Größe und auf den geringeren Hauptspeicher pro Einzelprozessor des Earth Simulators (4GB gegen 2GB).

Anhand dieses Beispiels ist gut zu erkennen, wie die Leistungsbewertung von real vorliegenden Systemen abstrahiert werden kann. Des Weiteren ist abzuleiten, dass Performance sehr von den an den Rechner gestellten Belastungsarten abhängt. Wenn diese also nicht exakt genug bestimmt werden, können Ergebnisse stark von der Realität abweichen.

5 Fazit / Ausblick

Abschließend lässt sich zusammenfassen, dass die Gründe für ein Scheitern bei vielen Software-Projekten das Fehlen einer frühzeitigen Ressourcen-Planung ist. Die Integration von Performance-Überlegungen in alle Phasen des Projekts, wie in Kapitel 1 gefordert, kann helfen, schwerwiegende und kostspielige Fehlentscheidungen zu verhindern.

Es wurden im Rahmen dieser Arbeit in Kapitel 2 die Grundlagen und in Kapitel 3 eine Reihe von Modellen für die Ressourcen-Dimensionierung aufgezeigt, die ein solches Scheitern verhindern können, indem die geplanten Systeme zuvor modelliert und ihre Leistungsgrößen gut abgeschätzt werden.

Durch das Beispiel in Kapitel 4 wurde aufgezeigt, dass eine solche Abschätzung vom realen Zugang zu dem zu untersuchenden System losgelöst ist. Es kann also auch ein noch nicht existierendes oder nicht zugängliches System modelliert und berechnet werden.

Durch die zunehmende Automatisierung der Modellerstellung, die in den nächsten Jahren auch noch weiter fortschreiten wird, werden diese Modellierungen auch immer leichter und damit kostenkünstiger. So können in Zukunft immer mehr Software-Projekte von einer erfolgreichen Ressourcen-Dimensionierung profitieren.

Literatur

Adco07. Matthew Adcock. Tis the season to scale up: Prepare your systems for the holiday rush, November 2007. Verfügbar unter: [http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1284172,00.html]; Abgerufen: 2008-06-04.

BCSS98. Marco Bernardo, Rance Cleaveland, Steve Sims und W. Stewart. TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems. In *FORTE*, 1998, S. 457–467.

Beck02. Klaus Becker. *Leistung und Performance von MVS-Großrechnern.* Oldenbourg, München; Wien. Juni 2002.

Beil81. Heinz Beilner. On the Construction of Computing System Simulators. In Domenico Ferrari und Massimo Spadoni (Hrsg.), *Experimental Computer Performance Evaluation: Lecture Notes of the Second Summer School on Computer Systems Performance Evaluation, Sogesta, Urbino, Italy June 16–27, 1980*, S. 1–31. North-Holland, Amsterdam; New York; Oxford, 1981.

BMIS04. S. Balsamo, A. Di Marco, P. Inverardi und M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, Band 30, 2004, S. 295–310.

BoRi89. Gunter Bolch und Helmut Riedel. *Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle*. Teubner, Stuttgart. 1989.

CNYM99. Lawrence Chung, Brian Nixon, Eric Yu und John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, Boston; Dordrecht; London. Oktober 1999.

Corp98. Georgia Tech Research Corporation. GVU's Tenth WWW User Survey Graphs, 1998. Verfügbar unter: [http://www-static.cc.gatech.edu/gvu/user_surveys/survey-1998-10/graphs/use/q11.htm]; Abgerufen: 2008-06-03.

Ferr78. Domenico Ferrari. *Computer Systems Performance Evaluation*. Prentice Hall. Mai 1978.

GiHi94. Stephen Gilmore und Jane Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the 7th International Conference on Computer Performance Evaluation*, 1994, S. 353–368.

Glas98. Robert L. Glass. *Software Runaways, Lessons Learned from Massive Software Project Failures*. Prentice Hall, Upper Saddle River, New Jersey, USA. 1998.

Grou03. Object Management Group. UML® Profile for Schedulability, Performance, and Time, version 1.0, September 2003. Verfügbar unter: [http://www.omg.org/docs/formal/03-09-01.pdf]; Abgerufen: 2008-06-01.

Grou05. Object Management Group. UML® Profile for Schedulability, Performance, and Time, version 1.1, Januar 2005. Verfügbar unter: [http://www.omg.org/docs/formal/05-01-02.pdf]; Abgerufen: 2008-06-01.

HaZo95. Martin Haas und Werner Zorn. *Methodische Leistungsanalyse von Rechensystemen*. Oldenbourg, München. 1995.

HeHK02. Holger Hermanns, Ulrich Herzog und Joost-Pieter Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, Band 274, März 2002, S. 43–87.

HeMo08. Frank Heitmann und Daniel Moldt. Petri Nets World: Petri Nets Tools Database Quick Overview, April 2008. Verfügbar unter: [http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html]; Abgerufen: 2008-06-04.

HHKM+00. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis und M. Siegle. Compositional Performance Modelling with the TIPPtool. *Performance Evaluation*, Band 39, Februar 2000, S. 5–35.

honl08. heise online. Bericht: SAPs neue Mittelstandssoftware „Business By Design“ klemmt. http://www.heise.de/newsticker/meldung/107116, April 2008. Verfügbar unter: [http://www.heise.de/newsticker/meldung/107116]; Abgerufen: 2008-07-10.

Jack57. James R. Jackson. Networks of Waiting Lines. *Operations Research*, Band 5, August 1957, S. 518–521.

KeHo03. Darren J. Kerbyson und Adolfy Hoisie. Sizing Equivalent Performing Large-Scale Systems Using Modeling. In *Innovative Architecture for Future Generation High-Performance Processors and Systems*, Juli 2003, S. 46–55.

Klei06. Torsten Kleinz. Weiter Wirbel um StudiVZ, November 2006. Verfügbar unter: [http://www.heise.de/newsticker/meldung/81562]; Abgerufen: 2008-06-03.

Klin05. Norbert Klingebiel. Service Level Management. *Controller Magazin*, 2005, S. 464–469.

Koun05. Samuel Kounev. *Performance Engineering of Distributed Component-Based Systems: Benchmarking, Modeling and Performance Prediction.* Shaker Verlag, Aachen. Dezember 2005.

Kozi04. Heiko Koziolek. *Empirische Bewertung von Performance-Analyseverfahren für Software-Architekturen.* Dissertation, Universität Oldenburg, Fakultät II, Department für Informatik, Oktober 2004.

LaCh87. Shui Fong Lam und K. Hung Chan. *Computer Capacity Planning: Theory and Practice.* Academic Press, Inc., Orlando, Florida, USA. April 1987.

Litt61. John D. C. Little. A Proof for the Queuing Formula: $L = \lambda W$. *Operations Research*, 9(3), Juni 1961, S. 383–387.

MeAD94. Daniel A. Menascé, Virgilio A. F. Almeida und Larry W. Dowdy. *Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems.* Prentice Hall PTR, Englewood Cliffs, New Jersey, USA. April 1994.

MeAD04. Daniel A. Menasce, Virgilio A. F. Almeida und Lawrence W. Dowdy. *Performance by Design: Computer Capacity Planning by Example.* Prentice Hall International, Upper Saddle River, New Jersey, USA. 2004.

MeAl00. Daniel A. Menasce und Virgilio A.F. Almeida. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning.* Prentice Hall PTR, Upper Saddle River, New Jersey, USA. Mai 2000.

Moll82. M. K. Molloy. Performance Analysis Using Stochastic Petri Nets. *IEEE Transactions on Computers*, Band 31, August 1982, S. 913–917.

NKPP+00. G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper und D. V. Wilcox. PACE — A Toolset for the Performance Prediction of Parallel and Distributed Systems. *The International Journal of High Performance Computing Applications*, Band 14, 2000, S. 228–251.

ReLa80. M. Reiser und S. S. Lavenberg. Mean-Value Analysis of Closed Multichain Queuing Networks. *Journal of the Association for Computing Machinery*, 27(2), April 1980, S. 313–322.

SaWP99. Greg Sandoval, Troy Wolverton und Jeff Pelline. Net retailers in eye of holiday storm, November 1999. Verfügbar unter: [http://news.cnet.com/2100-1017-233341.html]; Abgerufen: 2008-06-05.

Sche65. Allen L. Scherr. *An Analysis of Time-Shared Computer Systems.* MIT Press, Cambridge, USA. 1965.

Schw78. H. D. Schwetman. Hybrid Simulation Models of Computer Systems. *Communications of the ACM*, 21(9), September 1978, S. 718–723.

Sera86. Guiseppe Serazzi. *Workload Characterization of Computer Systems and Computer Networks.* Elsevier Science Inc., New York. 1986.

Walt06. René Walther. *Service Level Agreements.* Vdm Verlag Dr. Müller, Saarbrücken. Dezember 2006.

WaMR04. Ted J. Wasserman, Patrick Martin und Haider Rizvi. Sizing DB2 UDB® servers for business intelligence workloads. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, Markham, Ontario, Canada, 2004. IBM Press, S. 135–149.

Architecture Centric Model-Driven Software Development

Yanjun Zhang

Supervisor: Thomas Goldschmidt

Abstract Since the Unified Modeling Language(UML) became a popular tool in the software development field, the use of models has played a very important role in the developing process. But these models are usually treated more like documentation for the software rather than software itself, i.e. the source code. In the Model-Driven Software Development the models are considered just as the code, because the implementation of these models is automated through a key mechanism in this paradigm MDSD: the transformation. Architecture Centric Model-Driven Software Development(AC-MDSD) is a variant of the MDSD, where the architecture is a central point. This will greatly improve development efficiency, software quality and reusability.

1 Introduction and Motivation

1.1 Introduction

The software industry has grown into one of the strongest branches in the world economy in the last 40 years. But the search for an effective development method is still lasting. After the UML has come into the use, the softwares are much easy to understand than the pure code, but it is still not on the right way to achieve the efficient software development, because transformation of the models into source code is still software developer's tedious routine work. In order to achieve high development speed and better software quality, a promising approach is Model-Driven Software Development(MDSD). A proper definition of MDSD could be like:

> a software development approach, where models can be directly treated as the source code.

1.2 Motivation

As a variant of the MDSD, the central intention of the AC-MDSD is the automated implementation of the models and the reduction of the redundant code in the software development. That means in the mean time also increasing development efficiency, better software quality and reusability. In AC-MDSD the infrastructure code(relevant to software infrastructure) is created though the code generator. Thus the more formal the software architecture is, the more

schematic the source code is and the more code can be created with the code generator. Of course, there is no 100% generation, in fact, 60% to 80% of the software code can be generated from the architecture centric models, the software developers need to implement the domain specific part.

1.3 The Structure of This Report

The first part of this report is about the basic ideas and terminology in MDSD, especially the basic concepts in the Model-Driven Architecture(MDA) and the AC-MDSD are introduced.
The second part contains explicitly explained details about the elements in MDSD.
The third part aims at explaining an implementation in this paradigm AC-MDSD: Borland Together, and the contrast to other implementations is also discussed. The fourth part gives us a brief conclusion for the content of this report.

2 Concepts and Terminology in MDSD

2.1 The MDSD Approach

In this section I would like to introduce the basic concepts and ideas in MDSD and give an overview of two flavors for this paradigm: MDA and AC-MDSD. The keywords in this part are for example architectures, models, transformations, domain-specific languages etc. But first I want to cite a figure 1 from the book MModel-Driven Software Developmenttto illustrate and explain the basic ideas behind this approach.
If we analyze the code of an application(the upper left corner of the diagram), we can divide it into three parts(the lower left corner): a generic part that is identical for all future applications, a schematic part that is not identical for all applications, but possesses the same systematics(for example, based on the same design patterns) and an application-specific part that can not be generalized. At this point, we won't make any statements about the significance of each part: in extreme cases, the application-specific part can even be empty. MDSD aims at deriving the schematic part from an application model. Intermediate stages can occur during transformation, but in any case DSL, transformation and platform will constitute the key element. They must only be created once for each domain. [Thom06, p.15-16]

2.2 Common MDSD Concepts

In this section I want to introduce some important concepts in the MDSD using a MDSD-mindmap(see figure 2).

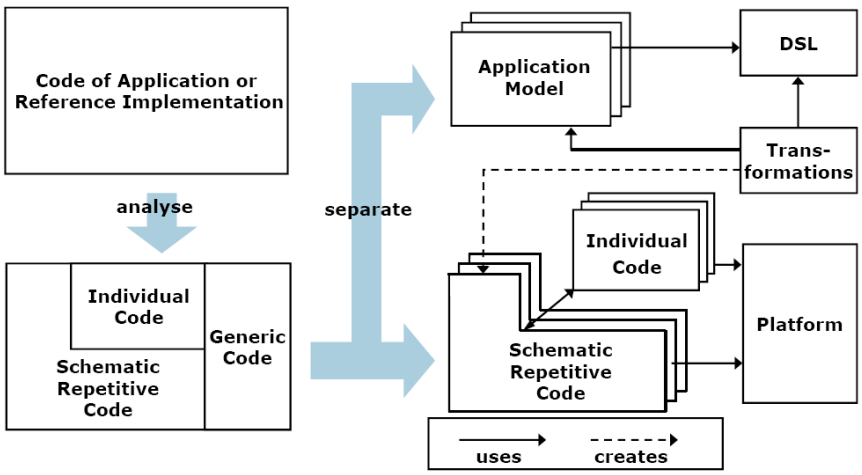


Figure 1. The basic ideas behind MDSD, Source:[Thom06, p.15]

Metamodels and Meta Meta Models: A metamodel describes how a model is constructed. Because the term mmetaïs relative, we can say that a metamodel is an instance of a meta meta model and a model is an instance of a metamodel.

Domains and Domain-Specific Languages: A domain is a special field, in which we are interested. We can also classify a domain in technical(like software development domains) or business(like financial or auto industry domains) domains.
Accordingly we can define a domain-specific language like this: a domain-specific language expresses important contents in a specific problem domain. A DSL in a technical domain can be textual(XML, HTML) or graphical(UML, BPMN); A DSL in a business domain can be special vocabularies, terminologies in a particular business field.

Abstract and Concrete Syntax: The abstract syntax specializes the structure of a language, whereas the concrete syntax of a language fixes what a parser for this language accepts. An abstract syntax of a language can have many different concrete syntaxes.

Static Semantics: The static semantics tell us what kind of rules must be obeyed in the language, how can the elements in this language well formed. For example, in the programming language C++, we can only declare a variable following this form: „variable-type variable-name;“.

Domain Architectures: The metamodel of a domain, a platform, and the corresponding transformations, including the implemented idioms, are the tools that

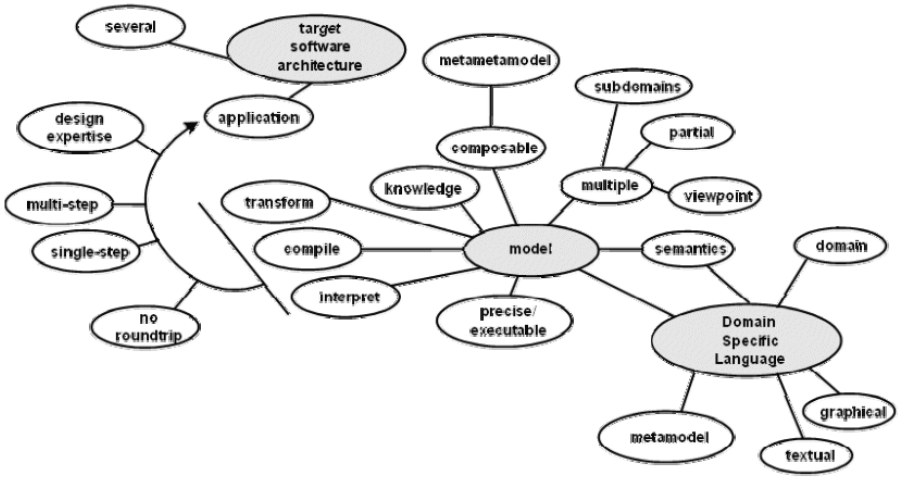


Figure 2. The MDSD Mindmap, Source:[Völta, p.4]

are needed to make the transition from model to the product, whether completely or partially automated. The aggregation of these items is what we generally call the domain architecture - the central MDSD concept. The generative software architecture discussed later is one example of a domain architecture.[Thom06, p.62]

2.3 The Goals of MDSD-Approach

The following items are those we are expecting from the MDSD-Approach: [Thom06, p.13-14]

- increases *development speed*. This can be reached through code generation from formal models.
- improves the *software quality*. The transformation from the models lead to an enhanced software quality.
- reaches a higher level of *reusability*. The defined architectures, modeling languages and transformations can be used to produce various software systems.
- helps to ease the *manageability of complexity through abstraction* using models.
- last but not least, it is necessary to refer the primary goals for OMG's MDA, it aims at improving *interoperability*(manufacture-independence) and *portability*(platform independence) of software systems.

2.4 The Concepts in Model Driven Architecture

Models: A model is an abstraction of an existing thing in the real world. In the context of the software system we can see models as an abstract representation

of a system's structure and behavior. In contrast to common UML models the semantics of DMA models are formally defined. This is because of the use of modeling languages.

Platforms: Although the abstraction level of platforms is not defined in MDA, but we consider a platform for software systems as a well-defined software architecture plus its runtime system. This is an important idea in the MDSD that will be specially discussed in later sections.

UML-Profiles: UML profiles are a standard mechanism to expand the UML vocabulary through stereotypes, tagged values and constraints. UML can be used in different domain and platform. A profile is a collection that contains these extensions which customize the UML models for special business or technical domains.

PIM and PSM: Domain-related specifications are fixed in Platform-Independent Models(PIMs) using a formal modeling language, mostly UML. These specifications are totally independent of the later implementation on the target platform(for example, CORBA, J2EE, .NET). Using automated model transformation, PIMs are transformed into Platform-Specific Models(PSMs) that contain the target platform's specific features. This is illustrated in the figure 3 .

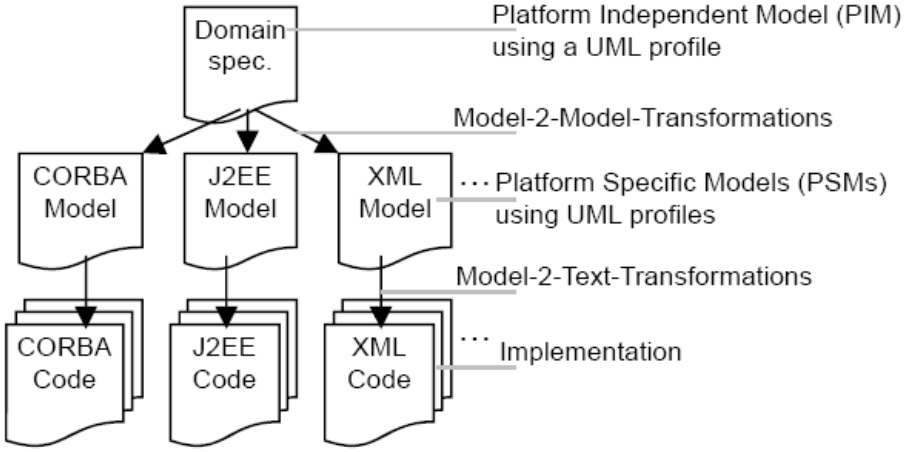


Figure 3. The basic principle of MDA, Source:[Thom06, p.16]

Transformations: Transformations are the core part of MDSD. This technique transforms models to their next level, i.e. models or source code. Therefore there are two types of transformation: model-to-model transformations and model-to-code transformations.

2.5 An Overview of AC-MDSD

First I want to cite a figure 4 to explain the principle of AC-MDSD. We see that in the center of the figure there is a generator template, which uses model-to-code transformations to transform the architecture-centric design model into the infrastructure code. In the upper part of this figure, we can see architecture-centric design models are created by using architecture-centric UML profiles. In the under part, the infrastructure code is supported by infrastructure components(platform), but the business logic code, which offers the domain-specific functionality, is created manually by developers.

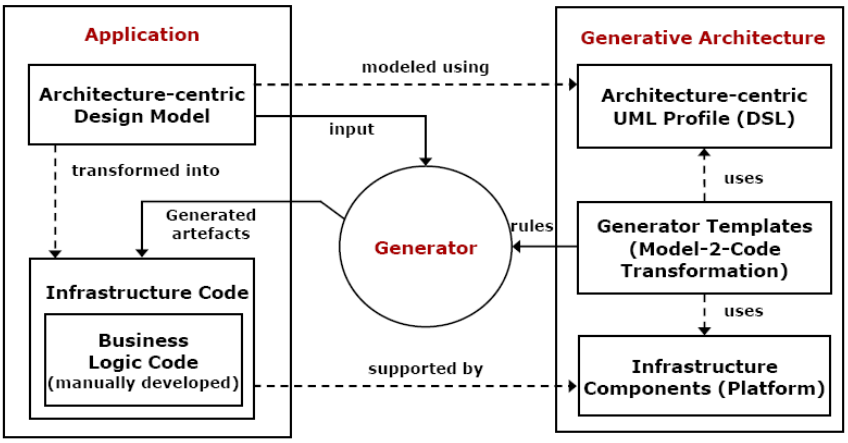


Figure 4. The principle of AC-MDSD, Source:[Thom06, p.23]

Generative Software Architectures: As we discussed before, in the AC-MDSD the software architecture plays a key role. If we create a better structured software architecture, we can use this architecture to achieve a more schematic source code. If the software developer write the completely self-thought-out source code, there will be very much work like copy and paste of the code, especially the code which represents the software architecture. If we let the code generator do this job, so comes the generative software architecture. In this case, the complete infrastructure code of the application is generated from the

architecture-centric design model through the code generator. But there is more, a generative software architecture support not just a single application, it can also support a family of architecturally-similar applications, a *software system family.*

The Properties of AC-MDSD: [Thom06, p.27-28]

- *Software system families instead of unique items.* AC-MDSD aims at creating generative software architectures for architecturally-similar applications.
- *Architecture-centric design.* In architecture-centric design we use platform-independent models.
- *Forward engineering.* Forward engineering means the transition from model to code.
- *Model-to-model transformation for modularization only.* A PIM should be as abstract as possible, but a direct transformation into the source code is still ideal. The ttransformation gapcan be modularized via model-to-model transformation.
- *No 100% generation.* Usually there is 60% to 80% of the source code can be generated from architecture-centric models. Of course the architectural infrastructure code is 100% generated, but the individual aspects must be implemented in the target language.
- *Software architecture becomes manageable.* Developers and designers can centrally detect the changes in the generative software architecture and handle them in the corresponding place, rather than in the entire software.

2.6 Comparisons

MDSD versus CASE: The development environments used in MDSD are not static, actually any target architectures, modeling and target languages and runtime components can be supported. This is the biggest difference from CASE. In a CASE tool, DSL, transformations, platform and target architecture are usually fixed.

AC-MDSD versus MDA: Here are some differences between these two flavors of MDSD listed.

- The MDA restricts on the using of UML-based modeling languages, AC-MDSD has no such restrictions.
- The primary goal of AC-MDSD is to increase software development speed, improve quality and reusability of software, but MDA improves the interoperability between different softwares through PIMs. AC-MSDS tries to diminish the gap between models and target platforms, so that we can create a platform, on which the most important architectural concepts are already to be used. MDA focuses on the possibility to generate different Implementation from a model for different platforms, and so - totally in the sense of OMG - to strengthen the neutrality of platforms of different softwares. [Völtb, p.3]

- In the AC-MDSD we use PIMs in architecture-centric design, but we do not use the PSMs in the designing process.
- In the AC-MDSD is the reverse engineering completely avoided. We want the architecture as abstract as possible, which makes no sense for models to derive changes from the source code.

3 Model-Driven Development

3.1 Models

Models can be textual or graphic, but they are usually considered as visual graphics which are designed using Unified Modeling Language or Business Process Modeling Notation(BPMN). The content of models can be enriched through declarative languages(for example, Object Constraint Language) as assertions.

UML and MOF: Unified Modeling Language(UML) is used to create a model for a real object, it is the second layer in the OMG's four metalevels. UML metamodel describes how to structure a model. Meta Object Facility(MOF) is a meta meta model, it is used to define the metamodel in M2-level. In order to understand the UML and MOF, I will use a figure 5 as an example to explain them.
We can see that an instance in the layer MO stands for a concrete element in a real world, in our example 6, a Car, has its color, owner, manufacture, price and ID. In the M1 it is a Class to model this instance, this class has a name: Car; has its attributes: color, owner, manufacture, price and ID; has the corresponding operations producedBy(car:Vehicle),buyACar(owner:Person), changeTheColor(color:Color), setAnID(car:Vehicle). In the M2 level there is a Classifier to construct the model in the next level, it has a name: Class and some features: attributes and operations. The fourth level: MOF's meta meta model is in this case also a Classifier and has a name classifier. In this figure, every level is a class of its underlying level, and an instance of its level above.

Meta Meta Models: As we have discussed before, MOF is a very popular used meta meta model, but I want to also list some other meta meta models.

- *Ecore.*
- *MetaGME.*
- *KernelMetaMetaModel(KM3).*
- *GOPRR(Graph, Object, Property, Relationship, Role).*

3.2 Architectures

Since architectures play a central roll in the AC-MDSD, we will discuss a lot about them in this part. First I want to cite a definition for software architecture:

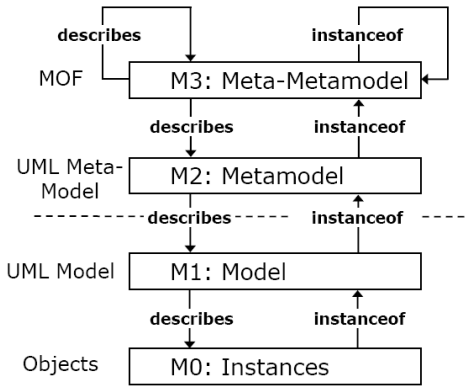


Figure 5. The four metalevels of OMG, Source:[Thom06, p.86]

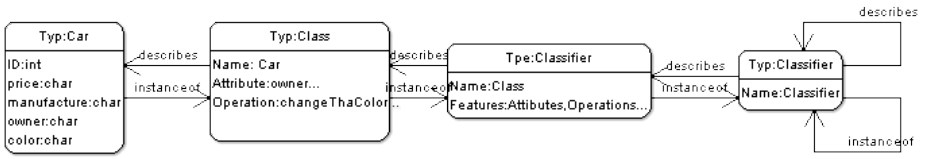


Figure 6. A example of four metalevels

Software architecture describes to a certain level of detail the structure(layering, modularization etc.) and the systematics(patterns, conventions etc.) of a software system. [Thom06, p.119]

A good software architecture must fulfil all the functional requirements(what kind of services can this software bring to us?) and expected non-functional requirements(usability, performance, space, reliability, portability etc.); It should be expandable under special additional requirements; A good-looking documentation helps here also. Regarding of AC-MSDS, the architecture must be as precise as possible and it should also support all the actual and potential product in a software family.

Target Architectures: A target architecture here means how the software product will be structured. This is a great concern in all kinds of software development paradigms. For the most software systems a layered model can be a very good reference. We can call this a five-tier architecture. At the bottom of this reference model is the operating system, which manages and coordinates the resources of a computer to accomplish different activities. The second layer is programming languages and libraries. These two layers build the basis for this

reference architecture. In the third layer there is middleware layer to provide basic technical services. These services include persistence, transactions, distribution, workflow management, GUIs, data management, scheduling or hardware access. We can also call this tier technical platform. For example in the business field .NET and J2EE are very popular technical platforms. In the fourth tier of this reference model is the business platform: this layer offers the basic concepts in the functional domain. These concepts include:

- *Entities:*This concept stands for a particular type of identity and its lifecycle. An example for it is a customer in a bank. He should has an identity, including all his information(account, job...) and a lifecycle, that means these informations must be stored in the bank-customer-management system as long as he banks here.
- *Value objects:*This concept does anything with values, an amount of money in a transaction, a position in a map-routing-system...
- *Constraints:*This concept gives the Entities the must-obeyed rules. For example, a driver can not be tested to have drunken alcohol in a traffic system. An employee can not be a children...
- *Services:*This concept defines the basic business services, for example the guarantee of persistence, the management of workflows, but in a particular domain.

At the top of this reference model for a target architecture is the application layer, which is a set of different softwares offering different functionalities.

Platform Architectures: Considering the MDSD paradigm, we can define a platform architecture, which is a part of target architecture. The MDSD platform is a very important concept, because the models have to be mapped to the platform to make them become executable source code. The process is realized through transformations. So the complexity of the transformation becomes a main concern in defining the boundaries of the MDSD platform in the target architecture. In order to decrease the complexity of the transformation we should make the concepts in the MDSD domain and the concepts in the MDSD as close as possible. There are several choices to reduce the complexity of the transformation. The following lists some of them. [Thom06, p.125-126]

- *MDSD domain and platform are located at the level of the reference model's technical platform.* The shortage of this option is its limited abstraction level, that means it can not exploit the full potential of the automation. But on the other hand, the MDSD domain is quite versatile - the domain architecture allows for the production of very different applications.
- *MDSD domain and platform are at the level of the target architecture's concepts.* AC-MDSD is an example for this choice. Here the architectural realization of the functional platform are derived from the reference model for the DSL definition as well as for the MDSD platform. The domain is less versatile, and the abstraction level is increased and a higher potential use of automation is achieved.

– *MDSD domain and platform are at the level of the functional platform of the reference architecture.* In this situation, the MDSD platform is the functional platform of the reference architecture. The domain is less versatile than in the architecture-centric case, and the full potential of automation can be 100% used.

How to divide the boundary of the MDSD platform from the target architecture depends on the type of domains. Usually the relative static entities, the properties of these entities and some constraints for the entities can be placed in the MDSD platform. In the context of AC-MDSD some parts of the technical platforms can also be some accessories for AC-MDSD plarform.

Domain Architectures: The domain architecture defines the basic part for all the products in a software system family. Building domain architecture includes modeling, choosing and building DSL, defining the platform architecture and the transformation architecture. The modeling and platform architecture we have discussed before and the transformation architecture will be introduced in the next part. We discuss here only the DSL construction.
The first step to build a DSL for a domain is choosing a proper DSL. An important criterion here is the needed variability in the domain. There is a good reference method to choose a DSL according to the Krzysztof Czarnecki's work. In his idea, DSLs are divided into two classes. One class provides only ways to do some routine configuration work. The other class can support creative construction. The DSLs in the first class provide a guidance for the developer and is particularly efficient. For Example, configuration parameters, property files, wizards; Tabular configurations can provide more freedom than the first three examples; Feature-Model based configurations is most powerful in the first class. If we have to do more creative construction work in the modeling, then DSLs providing graphical or textual languages can be good choices. The DSLs in this class are of course flexible, provide more freedom but in the meantime less guidance to the developer. Graph-like languages are good examples in this class. The other two alternatives are framework and manual programming, which provide the most freedom but the least guidance. Usually we use the combination of these two classes to describe a domain system.

Transformation Architectures: The main concerns in this part are for example, which part of the target architecture should be generated? What kind of the styles should the generated code have? Which concepts are very helpful in building a transformation architecture etc. All these concerns are useful to achieve an efficient transformation from models to the code. Let's take a detailed view of these aspects in the following part.

Which part of the code should be generated? Considering the target architecture, which part should be generated, which part should be manually programmed? In the context of a domain architecture, one will always generate those artifacts

that on one hand are not covered by the platform, and on the other, can not be described well and compactly using a DSL.[Thom06, p.150]

What kind of styles of the code does the transformation help? One reason why the generated code should be good-looking is that the developers have to insert the manually programmed part into the generated code to realize the domain-specific functionalities. So in building a transformation architecture we can generate sensible comments. Another very useful technique is the application of location strings which identify the templates used in the transformation and the model elements in the generated code.
There is another important style: Separation of generated and non-generated code. The biggest benefit of this choice is the avoidance of the consistency problem between models and the generated code in the modification of the generated code. The main reason for the inconsistency problem here is repeated regeneration of all generated artifacts. It offers a particular convenience in the regeneration process because the generated code can be simply deleted if it remains unmodified. If the generated code must be modified, then this change had better happen in the so-called protected area.

Which concepts are good suggestions for building transformation architecture? The first helpful suggestion is to exploit the model, we should use the information in the model as much as possible in order to minimize the manual implementation work by the developers. For example we can also generated configurations for platforms using the information contained in the model. These days we use often XML to specify the configurations. The second suggestion is that we should consider the system architecture into transformation architecture designing. For example, we can use the UML deployment diagrams to model software deployment and generate corresponding codes. The third concept is the integration of the MDSD system with the existing software systems. So in transformation architecture designing the integration aspects should also be considered. We can define a technical subdomain to support the definition of different mapping rules between model elements and existing applications.

Code generation techniques: Code generation techniques are a very important part in the transformation architecture, because without them a transformation into the source code is impossible. So next I will briefly introduce these techniques.[Thom06, p.186-199]

- *Templates + filtering:* This technique uses templates to iterate over the relevant parts of a textually represented model.
- *Templates + Metamodel:* This method implements first a parser for the textual model, then instantiates a metamodel, at last uses it with templates together for the generation.
- *Frame Processors:* In this alternative the frame is the most important element. It contains the basic specifications of code that will be generated. A frame can be instantiated for many times. The corresponding source code comes actually from these frame instances.

- *API-based Generators:* This is the most popular type of code generators. This mechanism uses an API to generate the elements in the target platform.
- *In-line Generations:* If you are familiar with the programming language C++, then I think you have already known some situations, where In-line generations are used. For Example, C++ preprocessor instructions or C++ templates.
- *Code Attributes:* This mechanism is very popular in Java. We know that JavaDoc is an automatic HTML-documentation generation. The extended architecture of JavaDoc has the ability to plug in the code generators.
- *Code Weaving:* This is a mechanism to intermix the separate pieces of code.

Tool Architectures: The tool architecture plays also a very important role in the MDSD. Today we usually find some integrated development environments, which contain different kinds of tools, including modeling tools, code generation tools, model-to-model transformation tools etc. We have to consider three basic aspects when designing a tool architecture. First, metamodeling functionality must be implemented in a tool so that we can check the correctness of the input models. Second, when we consider the transformation of the model in a tool, the concrete syntax of the model is usually ignored, because the definition of transformations based on the concrete syntax is simply too complex, so the transformations should base on the source metamodel. Third, we can build the modular transformations to enable the reuse of transformations or parts of transformations.

3.3 Tools

Model-Driven Software Development is supported by a variety of tools. According to different functionality, these tools can be categorized into 2 groups. The most important group is modeling and code generation tools, from relative simple graphical modeling tools like Microsoft Visio, Eclipse Family(Eclipse Modeling Framework, Eclipse Graphical Editor Framework, Eclipse Graphical Modeling Framework) to more complicated modeling tools such as IBM Rational Family and Borland Together. The second group is transformation and code generation tools, such like Pathfinder's PathMATE tool. In the following section I will introduce some detailed information about these tools.

Modeling and Code Generation Tools

Eclipse Modeling Framework: EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose, then imported into EMF. Most

important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications.[eclia]

Graphical Modeling Framework: GMF provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF. The project aims to provide these components, in addition to exemplary tools for select domain models which illustrate its capabilities. [eclib]Using it, we can draw a variety of diagrams, some of them are: BPMN Diagrams, ECore Diagrams, Mindmap Diagrams, UML2 Class Diagrams...

Microsoft DSL Tools: Microsoft DSL Tools are also modeling and code generation tools, including a wizard to generate models and a set of code generators. Let's take a look at this modeling environment 7 .

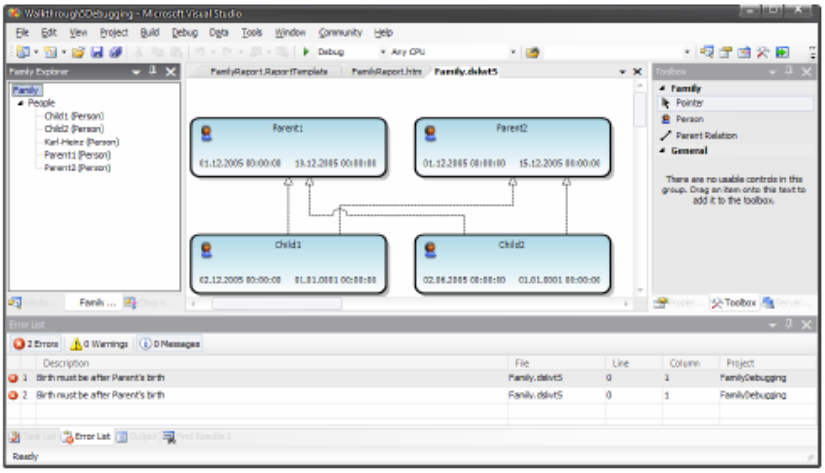


Figure 7. a screen shot of GMF environment

Borland Together: We will discuss this tool later in the fourth part of my report.

Transformation and Code Generation Tools

PathMATE Tool: The Pathfinder's PathMATE is the most open and configurable MDA transformation environment for the development of high performance software systems. It integrates the PathMATE automation and transformation technology and powerful UML modeling editors. The central part of this

solution is so-called PathMATE Model Automation and Transformation Environment, which consists of three blocks: the PathMATE Transformation Engine, which transforms the PIMs into software applications; the PathMATE Spotlight, a very advanced model-level testing environment and PathMATE Transformation Maps which drive the engine's transformation into different application systems running on different platforms.

3.4 Standards

The development of standards in the MDSD is essential for the realization of MDSD-goals. So next I will introduce some important standards in MDSD.

Modeling Standards: UML(including OCL) and MOF are two core concepts in the modeling standards and already introduced before. Here I want to introduce other three important standards in the modeling: Common Warehouse Management(CWM), Business Process Modeling Languages(BPML) and XML Metadata Interchange.

Business Process Modeling Languages: BPML is a metamodeling language for business processes. It bases on the XML, a metamodeling language for business data. BPML defines a formal model for expressing abstract and executable processes that address all aspects of enterprise business processes, including activities of varying complexity, transactions and their compensation, data management, concurrency, exception handling and operational semantics.[Arki02, p.1]

Common Warehouse Metamodel: The CWM is a modeling standard used to model metadata for the objects in data warehousing management, for example, relational, non-relational, multi-dimensional data objects etc.

XML Metadata Interchange: This standard is a mapping mechanism between MOF and XML is often used to serialize UML models in XML form. XMI is supported almost by all UML tools.

Transformation Standards: A very popular standard used in model-to-model transformation is Query/View/Transformation(QVT). This is a MDA technology. We know that the separation of PIMs and PSMs is a core characteristic in MDA. Before code generation happens, PIMs must be transformed into PSMs. But in the MDA standard it is not instructed how this step should happen. QVT is the very technological realization of this step.
The QVT standard specifies three different model-to-model transformation languages: two declarative languages: the Relations language and the Core language and one imperative language: the Operational Mappings language. This hybrid strategy can meet different requirements for different users.

Amount of source code[kB]	Traditional development	MDSD
Source code reference implementation	1.000	1.000
handwritten code	18.800	2.200
Models		3.400
Transformations		200
Total	19.800	7.800

Table 1. Code-volume ration in AC-MDSD, Source:[Thom06, p.369]

3.5 Processes and Engineering

MDSD Process: In the context of AC-MDSD it is suggested to separate the development between domain architecture and application in the development process. In the domain architecture development thread, building process should aim at producing software architecture systems with features like reusability, high quality and efficient. In the application development process the application developers work with the developed domain architecture. These two threads work with each other in an iterative style. That means in this iteration the application developers always work with the domain architecture from the last iteration. Another important concern in the development process is the target architecture development process, which aims to create a reasonable target architecture.

Testing and Version Management: Based on the common testing and versioning methods, in the context of MDSD there are special characteristics. For example, in testing there are two threads: tests in model-driven application development and domain architecture tests. In version management, in addition to the traditional versioning of application including models and in most case manually developed code, the generation tools, the generator configuration, the domain architecture and the MDSD platform must also be versioned.

3.6 Management

The topics in this part are for example decision support, quality management, investment and potential benefits, development team structure etc. Most of them are of economic aspects, which are not a big concern in my report. But one thing worth discussing here is some statistics about the benefits by using AC-MDSD. [Thom06, p.369-373]
AC-MDSD has a strong effect on the code volume. Here is a sample data from a strategic, web-based back-office application for a big financial service provider. We use a table to represent this sample data here.
From table 1 we can clearly see that the application of AC-MDSD concept can strongly reduce the artifact volume. Not only the code volume, the project effort can also be lowered greatly. The project effort usually contains works like analysis and documentation of the requirements, architecture designing, definition of modeling language, test generation and execution and project management.

4 AC-MDSD Paradigm Implementation with the example Borland Together

4.1 Overview of Borland Together's MDSD Technologies

Borland Together is enterprise-class software development environment. In Borland's Together exist many industry standards. Some of them are the model transformation standard QVT (Queries/Views/Transformations), the declarative language for describing rules in UML standards: OCL(Object Constraint Language), MOF(Meta-Object Facility) to define modeling languages. These standards are integrated in the Together Modeling Framework which is based on the Eclipse Modeling Framework. Next I want to list some technologies used by Together.[Fong06, p.4-5]

Modeling Transformation Technologies:

Mode-to-Model Transformation:

- Custom Model-to-Model Transformation via QVT.
- Model Merge, Copy and Compare.
- Bidirectional UML 2.0 to Data Modeling(Entity Relationship) Transformation.
- Bidirectional UML 2.0 to BPMN Transformation.

Model-to-Text Transformation:

- XSL Transformation (Supports Model Documentation Generation).
- Java Transformation (Supports Java Emitter Templates (JET)).
- Together Modeling Framework (TMF) and Eclipse Modeling Framework API support.
- Metamodel Browser.

Object Constraint Language (OCL):

- Model Quality Analysis (Model Audits and Metrics).
- Integrated Model Constraints Specifications.
- Full Featured OCL Expression Editor and OCL Expressions Viewer.

Query/View/Transformation (QVT):

- Implementation of OMG QVT Standard.
- QVT and Java Code Library Support.
- Full Featured QVT Editor, Debugger and Compiler.

Modeling Specification Technologies:

- Unified Modeling Language 2.0 (UML 2.0).
- Business Process Models (BPMN).
- Business Process Enterprise Language (BPEL).
- Data Modeling (ER Diagrams).

Common Together Technologies:

- UML Profiles (UML in Color and ER Logical Diagram Profile)
- LiveSource (Java and C++ Language Full Roundtrip Engineering)
- Together Patterns (Design Patterns and J2EE Patterns)
- Eclipse Plug-in Development (PDE) for Transformation Deployment

4.2 Comparison of Borland Together with other Software Companies' MDSD products

Considering the different software companies, MDSD capability is being added to their tool sets. These companies are not only those who traditionally focused on the middleware and development tools, for example IBM, Microsoft and Borland, but also those from the business application development side like SAP. Here I want to use a figure 8 from Forrester Research to illustrate the complete overview of the different MDSD features using by different software companies.[Giud07, p.13-14]

Borland's MDSD Implementation focuses on Tool Integration: As an active member in Object Management Group(OMG) and Open Systems Group(OSG), Borland's MDSD environment integrated a variety of tools to offer application development teams a broad choice of approaches.

IBM's MDSD is based on Open Standards and Open Source Software Eclipse: IBM is one of the founding members of Eclipse and the original donor of the Eclipse code base. So it is believed that the future of MDSD for IBM is still based on Eclipse and various OMG standards.

SAP's Commitment to MDD and OMG Standards brings New Ideas for the Business Application Development: SAP's Enterprise Service Architecture (ESA) strategy for providing a component-based platform is resulting in it delivering a full range of MDD products on its NetWeaver platform.

Microsoft takes its Own Approach to Model-Driven Development: Most of the Microsoft's concepts for MDSD is similar to those in OMG's MDA. The two most important concepts in the Microsoft's implementation for MDSD is its DSL and software factories, which are configured in a software development environment to efficiently create a specific kind of application.

5 Conclusion

AC-MDSD provides an advanced engineering principle in the software development field. In this paradigm the building and designing software architectures is a very important part in the software development process. As a major

	Microsoft	IBM	SAP	Borland
Model-driven development (MDD) approach	• Several domain-specific languages, for design of apps, systems, workflows, and others • Supports creation of custom DSLs • Broader strategy: software factories • Integrated in Visual Studio 2005	• Based on open standards: MDA (UML and BPMN) and IDEF • Partners provide additional capability (SysML) • Built around Eclipse	• Based on MDA (UML and MOF) and BPMN • Supports iterative MDD, mixed MDD*, graphical modeling, design of transformations, messages, and cross-component integration • Range of tools target application and infrastructure developers and business process experts	• Based on MDA, QVT, and OSG • Family of products all based on same architecture • Varying levels of functionality, for various individuals
MDD tools:	• Visual Studio 2005 • Pricing: see Microsoft Web site • Tools are available for free in Visual Studio SDK	• IBM Rational Suite • WebSphere Business Modeler • WebSphere Integration Developer	• Web Dynpro (SAP NetWeaver Developer Studio) • Together UML (SAP NetWeaver Developer Studio) • Aris (SAP NetWeaver Developer Studio) • SAP NetWeaver Visual Composer • SAP Composite Application Framework • SAP NetWeaver Exchange Infrastructure	• Together 2006, Eclipse release 2
Model and metamodels	• XML-based format • Interactive language designer • Numerous options for code customization	• Underlying metamodel is MOF (UML Support) and EMF (Eclipse-based tools) • Supports UML, BPMN, and SysML	• Modeling infrastructure approach • Based on MOF and uses XMI and JMI to define MOF's XML and Java bindings • Builds scalable foundation for integrating various modeling and MDD tools • UML is in SAP's MDD picture via Together UML for SAP NetWeaver	• Together supports multiple types of modeling (UML, PEL, and MOF) • UML 1.4 and UML 2.0 • BPMN • ER Notations • Eclipse EMF and GMF • QVT for model transformation • Model serialization in XML through XMI
Programming languages and platforms	• Open text templating • Can generate textual artifacts of any kind	• Java/J2EE, C/C++, and CORBA • Supported on Windows and Linux • Rational Rose: different version for Java, VS, Data Modeler, etc.	• Java and ABAP • Interoperates with .NET and other MS environments and languages	• Java, C++, C#, .NET, CORBA, WSDL, and BPEL

Figure 8. The features of different MDSD implementations from software companies , Source:Forrester Research, Inc.

flavor of MDSD, it aims at increasing development efficiency, software quality and reusability. The AC-MDSD has the goal of generating a variety of software(software family) instead of generating the same software for different platforms. The motivation of this paradigm comes from the observation that the infrastructure code does have a considerable part of the entire source code. For example, in e-Business applications, this part lies around 70%, but in applications with programming closer to hardware, this part lies even more, for instance in embedded system development, this share lies usually between 90% and 100%. The AC-MDSD is also ideally suited to provide modularization constructs that can already be applied during the elaboration of requirements and that can automatically be enforced in the form of a strictly component based design within the implementation code. The prospect of being able to actively manage and control dependencies often is the key motivating factor to employ architecture-centric MDSD.[Bett06, p.10].
But the AC-MDSD has also its disadvantages, for example, the companies, which want to apply this principle, must endure high investments at the beginning. These investments include development of a DSL, modeling tools and generators, education employees etc, these tools are usually not mature enough. At last, the AC-MDSD is still a relative new technology compared with many other software development paradigms and does not have so much experience. Despite of its high investments at the beginning, AC-MDSD is still considered as one of the most promising software development technologies in the future.

References

Arki02. Assaf Arkin (Hrsg.). Business Process Modeling Language. Technischer Bericht, Intalio, November 2002.

Bett06. Jorn Bettin (Hrsg.). Transitioning to Model Driven Software Development - Preparing for the Paradigm Shift. Technischer Bericht, SoftMetaWare Ltd., April 2006.

eclia. eclipse.org. www.eclipse.org/modeling/emf/.

eclib. eclipse.org. www.eclipse.org/modeling/gmf/.

Fong06. Choong Koon Fong (Hrsg.). Quick Start Guide to MDA - A Primer to Model-Driven Architecture, Using Borland Together Technologies. A borland white paper, http://www.borland.com, Together Product Champion - APAC Support Center, September 2006.

Giud07. Diego Lo Giudice (Hrsg.). The State Of Model-Driven Development. Forrester research, http://www.borland.com, April 2007.

Thom06. Markus Völter Thomas Stahl. *Model-Driven Software Development.* Wesley. 2006.

Völta. Markus Völter. Modellgetriebene Softwareentwicklung. *ingenieurbüro für softwaretechnologie, www.voelter.de*, S. 3.

Völtb. Markus Völter. Sprachen, Modelle, Fabriken. *ingenieurbüro für softwaretechnologie, www.voelter.de*, S. 2–3.

Software Extension Mechanisms

Benjamin Klatt

Chair for Software Design and Quality (SDQ)
Institute for Program Structures and Data Organization (IPD)
University of Karlsruhe, Germany
Adviser: Klaus Krogmann
14.07.2008

Abstract Industrial software projects whether in-house or open source not only have to deal with the number of features in the system. Issues like quality, flexibility, reusability, extensibility and not at least developer and user acceptance are key factors in these days. An architecture paradigm supporting these issues are extension mechanisms. The main contribution of this paper is to identify characteristics of software extension mechanisms. Furthermore it presents different extension mechanisms and gives an overview on important aspects when introducing an extension mechanisms in own software.

1 Introduction

In 1979 Parnas wrote about the advantages of extensible software [Parn78] and why developers should start to adopt to this. He already named typical problems of software engineering like:

- Delivering an early release with a subset of features.
- Adding simple features without enormous code changes.
- Removing unneeded functionalities as product variants.

These are only examples for the need of developing flexible and reusable software in a fast and reliable way. In the last years, a lot of research has been done in this area. Object Oriented Programming (OOP), Component Based Development, Service Oriented Architecture (SOA), Model Driven Software Development (MDSD) and a lot more have influenced the software development discipline. Extension mechanisms are neither a hype like some of them nor a new fundamental innovation. They exist since a long time and they are more a design decision and a combination of other existing technics. Different mechanisms can vary a lot only sharing the intention to extend a software.

On the one hand, providing a good extension mechanism can lead to a successful and widely accepted product like some of the examples given in this paper. On the other hand, a bad one can constrict the evolution of a software, produce a lot of costs for maintenance and support and lead to rejection by developers and users. Up to now, there is no paper directly dealing with the comprehensive issues of extension mechanisms in a product independent way.

This paper uses three key terms. An “extension point” is the definition of the provided interface for extensions. An “extension” itself is an implementation according to an extension point. And an “extension mechanism” includes everything about an explicit extensible part of a software.

The contribution of this paper is the definition of different characteristics of software extension mechanisms. These characteristics are a result of an analysis of existing solutions. Additionally, this paper gives examples of existing software with extension mechanisms and gives guidelines for introducing them in an own product.

1.1 Related Work

As stated in the introduction no paper has been found that directly deals with extension mechanisms but a lot of work has been done strongly connected to this topic.

Extensible software in general Parnas was one of the first talking about extensible software [Parn78]. He stated why extension mechanisms are required, what the challenges and what the advantages are. He introduced a lot of design principals to consider. Well designed extension mechanisms are an application of his ideas and add additional results from the research of the last years.

API / SPI Design The design of Application Programming Interfaces (API) and Service Programming Interfaces (SPI) is one of the key factors for good extension mechanisms. They define how extensions execute a part of a system like a library (API) or how they are executed by the system in case of a framework (SPI), respectively. In this field a lot of work has been done. Joshua Bloch has summarized a good set of principles for API design [Bloc06].

Framework Design Frameworks have been developed as a result of the goal to simplify repeated work and to support a short time-to-market development. They combine generalized code and components, best practices, and SPI design. There is no standardized specification for the term “framework” but in this paper we separate frameworks from libraries by the fact that frameworks provide a basic application that is the main executed part of the system. Libraries are called and not the leading part in the system. Extension mechanisms are somehow included in most of the existing frameworks and thus a lot of experience can be gathered from the work done in this field.

Component Based Software Development Component based software development [SzGM02] deals with a lot of issues also targeted by software extension mechanisms. Both can be seen as a modification of the other one as long as the extensions are well encapsulated. The extensions developed for a specific software can be treated as components. A software providing an extension

mechanism can be handled as a concrete component based system. Often, the extension mechanisms are provided by component frameworks like Eclipse RCP or Enterprise Java Beans (EJB) but the design of extension mechanisms is not necessarily related with components.

Software Factories The concept of software factories is to use state-of-the-art development technics to develop a modularized software that can be assembled according to the customer specific requirements. The goal is to optimize the development life cycle and the reuse of existing components. Jack Greenfield is one of the most present researchers in this field and has written a book [GrSh04] about this concept. Extension mechanisms can be used as the foundation for customer specific software assemblies.

Product Specific Extension Mechanisms Extension mechanisms are not a new idea and every software that can be extended not only by changing its code provides something like an extension mechanism. Most of them are proprietary and having a wide range of quality. Even if they are good or bad they can provide a lot of experience for how to design an extension mechanism. In this paper there are four different products used as examples: OSGi [OSGi08b], Eclipse RCP [Ecli08a], Typo3 [Typo08a] and osCommerce [osCo08c] whose characteristics will be evaluated in a later section of this paper.

1.2 Outline

The remainder of this paper is structured as follows: In Section 2 we will discuss different characteristics of extension mechanisms. Section 3 presents examples of software with extension mechanisms of different types and quality. Section 4 discusses issues of introducing an extension mechanism in your own software and Section 5 gives a conclusion and sums up the results of this paper.

2 Characteristics

This section describes the characteristics of extension mechanisms identified by the analyses done for this paper. The characteristics can be specific to an extension mechanism as well as reasonable for a whole software development process. They will be discussed for both aspects.

Feature diagrams as defined by Czarnecki and Eisenecker [CzEi02] have been developed for this paper to visualize the characteristics. Figure 2 gives a top level overview of the characteristics. The feature diagrams can be read according to the legend in Figure 1.

Which of the characteristics are important depends on the goal of the extension mechanism in the specific software. However all of them should be considered when designing a software with an extension mechanism. How this can be done is discussed in the later sections.

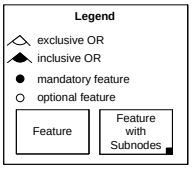


Figure 1. Feature Diagram Legend

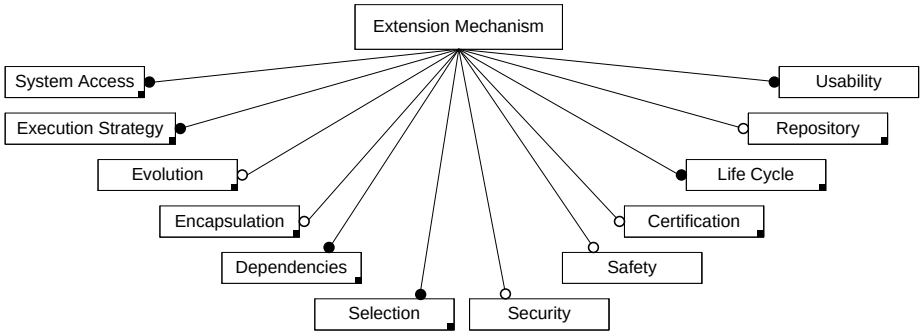


Figure 2. Top Level Characteristics

2.1 System Access

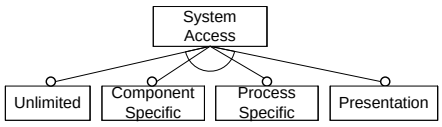


Figure 3. System Access

A major decision when designing a software with an extension mechanism is the system access for the extensions. As shown in Figure 3 this can reach from user interface extensions on the presentation layer to the point of the system core. Additionally, extensions can be specific to a component or to a business process. While the first one is more connected to single software components the second one can be component comprehensive and may not be directly connected to software aspects.

Possible is any of these but it has to be clarified what are the consequences for the target group and its requirements. The deeper the extension mechanism is introduced into the system the more flexibly the software can be changed, but also issues like complexity, security and reliability have to be handled more carefully.

2.2 Execution Strategy

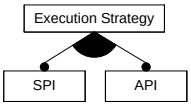


Figure 4. Execution Strategy

There are two different execution strategies for extension mechanisms. Either the extension is executed by the system or the extension executes the system or at least a part of it. The former one is the case for frameworks where the framework represents a base application that calls the extension. The latter one is more related to libraries where the extension executes the rest of the application and triggers existing functionality such as libraries. This is like a user interface or scheduler as extensions that are executing the application. Those two strategies go in parallel with SPIs and APIs (Figure 4). A framework makes use of extensions implemented according to Service Programming Interface. Libraries provide an Application Programming Interface and they represent themselves components or extensions which can be executed. It is important to distinguish between those two to know how they can be evolved, deployed and which of them is the controlling part in the system. For example it is possible to remove called methods from a SPI and existing extensions can still be executed. For APIs on the other hand, it makes no problem to add new methods but it makes problems removing methods from the interface.

2.3 Evolution

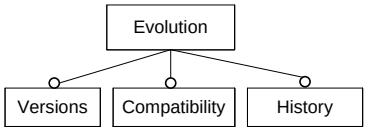


Figure 5. Evolution

The evolution of extensions as well as of the extension mechanism itself is as hard as with every open system. Figure 5 presents key factors for the evolution. The mechanism has to be as stable as possible or at least backward compatible. Otherwise all existing extensions will not work anymore. At this point the same evolution rules as for APIs and SPIs have to be taken into account. Adding functionalities to an API or removing some from an SPI should not change the execution behavior or the feasibility of an extension. But the other way

around, to add required calls to an SPI or remove some from an API should be avoided to ensure backward compatibility. Otherwise migration support has to be provided. The extension dependencies section discusses another aspect that is tightly connected to the evolution of extensions. Managing the extension history becomes necessary if it is required to get back to an older version, e.g. if an invalid new extension was installed or a product bundle with an older functionality set should be packaged. The extension history is also necessary if the mechanism has to be able to support parallel instances of different versions of the same extension, e.g. for differing extension dependencies. Furthermore, the history provides a good feedback about the extension development activities.

2.4 Encapsulation

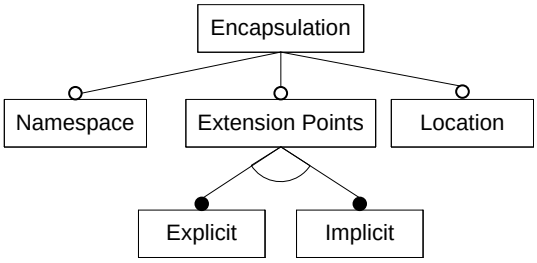


Figure 6. Encapsulation

The encapsulation is a characteristic to separate different extensions from each other and/or from the main software providing the extension mechanism. In Figure 6 the three aspects namespace, extension points and location are visualized.

Namespaces are used for the identification of extension artifacts. This includes objects as well as configurations or any other elements that should be identified as connected to an extension.

Extension points represent a well defined interface between the extension and the extended system or other extensions. This can also support the quality for all of them by clear interfaces. For example it makes it possible to provide a test framework that implements an extension mechanism with the capability to test the extensions without the whole extended software in place. The other way around, special test extensions can be developed and loaded in the system to test the system itself. With a good design this can be used for unit as well as for (pre-) integration tests. The clear separation and testability without the complete system in place supports the development, too. If the extensions have a good specification, they can be allocated to different teams to speed-up the development. Furthermore this gets important for packaging releases with a subset of functionality. This can be used either for early releases or

for reduced variants in conjunction with product line development [GrSh04]. Extension mechanisms can provide facilities to describe those extension points explicitly or they can be described informal e.g. within a system documentation.

The location characteristic is important to identify the resources of a specific extension.

Without any of these features, extensions are just direct code manipulations in the core software.

2.5 Dependencies

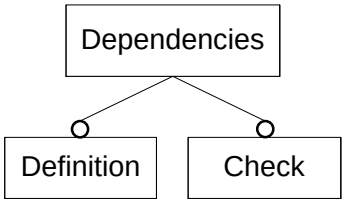


Figure 7. Dependencies

A hard to handle issue are the dependencies between extensions. For example extensions may need to access the same resources. Also it is not only possible to design an extensible software, but also to design extensible extensions. This brings in a lot of flexibility but complexity as well. Extensions will change independently. It is important to take care for this dependencies to not getting to complex or impossible to be resolved. This can happen if two extensions A and B require different versions of a third extension C but the software supports only one instance for each extension. This may result in an irresolvable problem.

To handle this two basic facilities, making these dependencies explicit are is (Figure 7). Dependencies definition makes it possible to better identify them for the user and automate handling for the system. Check of these dependencies by the mechanism is an additional feature making it possible to handle more complex dependencies even with more than two extensions involved.

2.6 Selection

Selecting a component implementing an extension can either be completely manual or supported by the system (Figure 8). The selection consists of two major steps. First, potential candidates have to be found, afterwards the best one of them has to be chosen. This can be done completely manual or supported by a system. The manual way depends on the person who wants to select the extension. With system support a search can be provided for the identification of potential candidates. This can range from a simple keyword search to a semantic or context sensitive one. To select the best candidate a feedback system

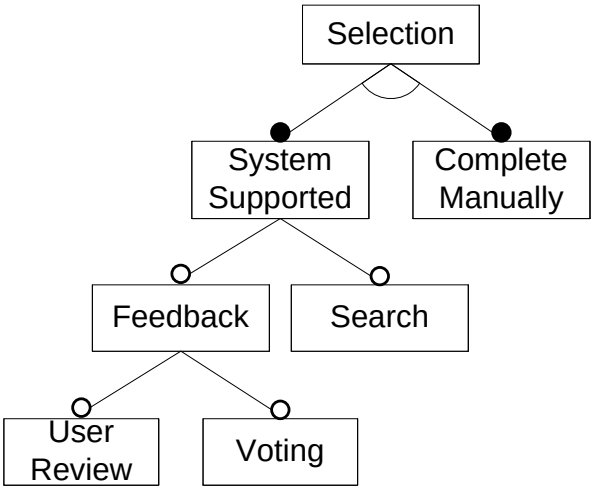


Figure 8. Selection

can be implemented. For example voting mechanisms or user reviews can provide information to support a decision. It is important to take functional as well as non-functional requirements into account. This is also tightly connected to the certification and repository characteristics of the extension mechanism. [AlLC07]

2.7 Safety

The safety is about the protection of the system. It is a design challenge to make a system-crash impossible by the use of an extension. Also to keep the overall reliability independent from the extension has to be defined by the architecture of the extension mechanism.

The main factors for the safety are: which parts of the system does an extension have access to and who can deploy or access the extension. The first issue is about which impact the extension might have on the system. It is closely related to the system access characteristic but with another point of view. The latter one decides whether this impact will be taken into account or not.

An extension with deeper access to the system will be of higher risk if everyone can manage it rather than only a limited and well trained group of users can do this.

2.8 Security

While the safety handles the system protection, security is about the data protection. It is influenced by the data the extension provides access to and the rights to whom it provides this access. It has to be ensured that only those

people have access to the data who are intended to. A more sophisticated way is to provide a configuration for the data access.

2.9 Certification

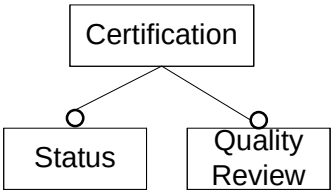


Figure 9. Certification

The evaluation of candidates for a required extension may become an expensive or at least a time consuming task. Certifications can provide some metrics of quality attributes and well defined specifications to support this evaluation process. Certificates are not only a technical aspect. If business requires selling extensions, providing quality guarantees is likely to increase revenues.

Figure 9 shows two aspects of the certification. One is the status of the extension. Which identifies the maturity of the extension and provides information on whether it can be used for example in productive systems. Another aspect which can be found in certifications are reviews. Those reviews have to be performed according to a standard process but can provide a trustworthy feedback and quality assurance. Moreover a manual review will lead to personal signature by the reviewer.

The drawback of certifications is the effort of the certification specification itself as well as the certification of the individual extensions. This effort makes especially sense for very often reused or commercial extensions. [AlLC07]

2.10 Life Cycle

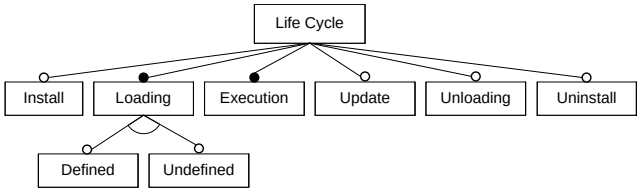


Figure 10. Life Cycle

Which states an extension can run through is defined by the life cycle introduced in the extension mechanism.

The states that extensions run through do not have to be explicit in an system but can be identified as follows:

install The physical extension resources are brought into the system.
loading The required resources and configurations are loaded by the active system instance. They are analyzed and initialization steps might be performed.
execution The real execution of the extension. This combines start, run and stop of processes.
update Any changes of the extension configuration or resources are performed.
unloading The resources and configurations are removed from the active system instance.
uninstall The physical extension resources are removed from the system.

The point in time where extensions are loaded can affect the start-up time of the system, the first execution time and/or each execution of an extension. It has to be clarified whether the start-up and reboot, the reaction time or for example a check with each execution are important or not. Depending on the implementation of an extension the loading strategy can influence the functional behavior, too. For example if an extension estimates being loaded once at system start-up time and prepares some global resources. It can run into trouble if it is reloaded with every execution and tries to prepare those global resources over and over again. It is important to decide that extensions either have to be independent from the loading time or to define when the loading will happen.

2.11 Repository

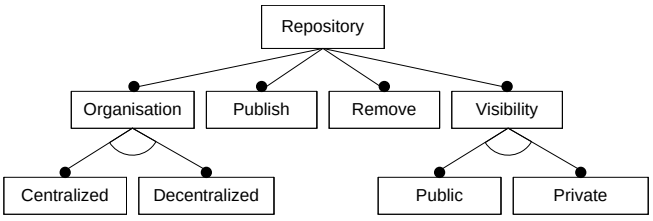


Figure 11. Repository

Writing extensions not only for designing flexible software, but also for providing reusable components, requires a place to store those components. Such a place is called an extension repository.

Depending on the extensions and the designed mechanism, the repository can range from a simple hard disc directory to a web based, deployment supporting service point. At least it has to support publish and remove actions.

Specifying a repository requires to define its organization. It can be built either centralized or decentralized according to the typical advantages and disadvantages of creating a bottleneck or loosing control of a consistent structure. Another aspect is the visibility of the whole repository or several extensions. Depending on the requirements it might makes sense to introduce a separation of public and private visibility.

A well designed repository is also influenced by other characteristics identified in this paper, like

- version control
- selection
- life cycle
- certification
- and some others

2.12 Usability

Independent from the goal of an extension mechanism it is important to design it to be usable as much intuitively as possible. If developers can use it with a clear understanding without a huge training effort, this will support a broader acceptance and use. As with the API and SPI design extensions, they have to be as easy to understand as possible. The already mentioned overview of principals on interface design given by Josha Bloch [Bloc06] also applies to extension mechanisms.

3 Existing Extension Mechanisms

Nearly all software products out in the market, which are not individual developments for a specific closed user group, provide the possibility to be extended. For this paper we have selected a group of open source products representing a large variety of the above characteristics, which are widespread in the market.

3.1 Open Services Gateway initiative (OSGi)

OSGi is developed by the OSGi Alliance [OSGi08b] and provides a dynamic module system for Java. The so called OSGi Service Platform is not a product with an extension mechanism but an extension mechanism itself that can be integrated into other products. It provides a definition of small components and offers their collaboration, dynamical loading and updating as well as a set of standard component specifications (Figure 12).

OSGi already takes care of a lot of characteristics stated in this paper. It explicitly encapsulates the extensions by namespaces and explicit extension points as well as the location of their sources. It also adds private classes to the Java environment. This limits the access to the intended extension points of an application. Private classes lead to a more secure main application by preventing to bypass the extension points.

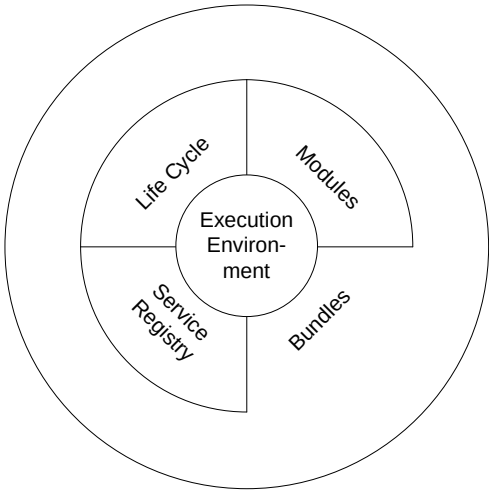


Figure 12. OSGi Framework Design [OSGi08a]

OSGi supports an explicit life cycle and a service registry that works together with extension repositories.

The OSGi platform is a base for a flexible software development and thus a good base for extension mechanisms. Nevertheless, when developing a software based on OSGi it is still required to take care of all described characteristics to design it with a good extension mechanism.

3.2 Eclipse RCP

Eclipse does not only provide one of the most successful Integrated Development Environments (IDE), but also a Rich Client Platform (RCP) which is the base for the IDE itself. It is a Java-based open source software. One part of this software is the Equinox Framework [Equi08] which is an implementation of the OSGi specification. It is placed in the small core of the platform and everything is build as plug-ins around it. In the Eclipse terminology plug-ins are what this paper calls extensions.

Figure 13 shows the basic Eclipse architecture. The base of eclipse is its runtime. Equinox, the OSGi framework is based on this runtime. Everything else is build on top of it as plug-ins. The first level is the Eclipse Rich Client Platform update and UI components. The UI component includes the SWT (Standard Widget Toolkit) and JFace which provides advanced SWT components. The next level which includes plug-ins developed based on the Rich Client Platform is the Eclipse Platform providing platform service plug-ins like search, help, launch and debug facilities and a lot more. On top of this layer different projects like the Eclipse Tools or the Eclipse Software Development Kits (SDK) are built. Those projects provide subprojects like the Eclipse Modeling Framework (EMF),

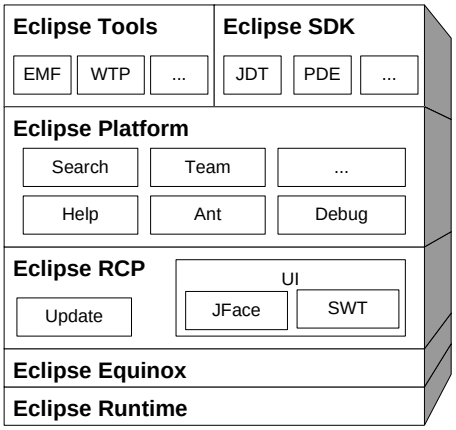


Figure 13. Eclipse Architecture [Ecli08a]

the Web Toolkit Platform (WTP), the Java Development Tools (JDT) or Plug-in Development Environment (PDE). This architecture shows that everything above the Equinox Framework is realized as plug-ins.

On top of the Equinox OSGi core a lot of the characteristics described in this paper are handled by the Eclipse RCP in an advanced way. The platform provides explicit extension points. They are not only programmatically defined as public but also defined in the plug in descriptor. Those extension points can be declared for extending extensions as well. The RCP can be reduced to a small core to develop really light-weight applications. To make this possible, extensions can be developed with nearly unlimited system access.

Eclipse RCP also provides a mechanism called update sites. Those are web-based decentralized extension repositories which can be provided in public and private environments, too. The public and private separation can only be done on update site and not on extension level. In the last years a couple of community websites like Eclipse Plug-in Central [Ecli08b] have been built to provide an extension directory with links to the extensions and their specific update sites. This community websites also provide search and feedback functionalities.

Eclipse inherits the advanced extension life cycle from the OSGi specification. The most important point is that extensions will be scanned for their descriptors when the system starts up, but the real loading is only performed when the extensions are executed for the first time. This enables installing a lot of extension but still providing a short system start-up time.

The dependency handling is done from OSGi, too. With the extension descriptors it is possible to define dependencies and also to handle different extension versions. A check facility is also available in the Rich Client Platform. It is able to look up for missing referenced extensions or to give a feedback if there is a critical extension setup.

The execution strategy is not limited to APIs or SPIs. It is not only possible to let the application execute and control an extension, but also to build an extension that fires events to the core application.

Over all, the Eclipse RCP is one of the most advanced bases to provide extension mechanisms within a software. But the characteristics presented in this paper still have to be considered using this base in the right way.

3.3 Typo3

Typo3 [Typo08a] is one of the leading open source Content Management Systems (CMS). It is developed with PHP and provides a flexible database connection.

One of the key factors for the success of Typo3 is the clean extension mechanism. As shown in Figure 14 the system has an extension API on top of it's core. Both in the frontend and in the backend, all functionalities are provided by extensions developed against this unified extension API. In Typo3 extension points are called extension hooks. Because Typo3 has a strong focus on safety and security, those hooks provide a lot of flexibility but ensure that it becomes hard to crash the system completely. Extension hooks can be defined in the extensions, too, to make them extensible again. The Typo3 extension API not only provides those extension points but also a lot of utility code as well.

Extensions can be published in the official repository called TER (Typo3 Extension Repository). Each Typo3 installation that is connected to the internet has an integrated mechanism to download extensions from this repository and install them automatically. It is also possible to upload extensions directly in an installation. Each extension is identified by an extension key [Typo08b]. If the extension is made public in the TER it has to register a unique key first. This key is not only used for identification but also provides a namespace to encapsulate extensions from each other. Typo3 contains an extension manager to install, update and remove extensions. It is possible to chose any of the published versions of an extension to be installed but only one specific version at a time. At the 28th May of 2008, the TER contained 3,200 official published extensions.

The search for extensions is supported by the web frontend of the extension repository. It provides a free text search and lists for different criteria like popularity or the last update date. The extension selection is supported with a tight connection to the quality assurance. Each extension has to match a specified structure. If this is fulfilled a manual is generated automatically out of the extension and linked in the web interface. There are also download statistics and member voting as a feedback about the extensions. Furthermore Typo3 defines four extension development stages:

- Experimental
- Alpha
- Beta
- Stable

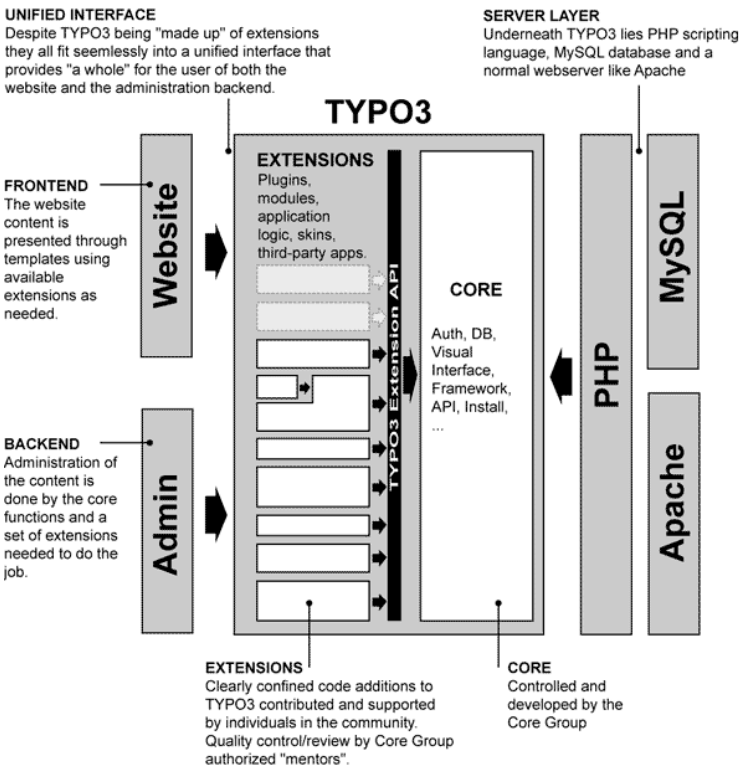


Figure 14. Typo3 Architecture [Skaa07]

The development state is defined by the developer itself but rarely misused especially for stable extensions.

One really strong quality assurance are the reviews done by the Typo3 Security Team. They started in 2006 to perform extension reviews for security issues [Typo08c]. In the last month not very much work has been done at this point but a list of already reviewed extensions is available on the repositories web interface.

As stated before, there is a well defined structure for extensions and also a template for documentation. If this is used, the documentation can be accessed within the extension manager of each Typo3 installation as well as downloaded or viewed as HTML in the web interface of the repository.

Typo3 also takes care of the evolution of extensions. Within the extensions description a version number can be maintained. Which can support inter extension references. Thus, other extensions can reference to a specific version of the extension. Furthermore, this supports an update mechanism within the extension manager of Typo3. The version number is also taken into account for minor things like the download statistics of the extension repository.

One thing that is left open is the loading strategy. Due to the PHP environment, in general the extension code is loaded and executed on demand. Depending on the PHP interpreter some parts of the program might be cached. Typo3 has introduced an installation sequence for extensions which is designed to set up the required extension environment. An extension is imported, installed and loaded. The import is only for getting the resources in the installation. The install step is for preparing the environment like building required directory structures or database tables. And the loading step is for activating the extension for the system.

The Typo3 community has invested a lot of effort in the extension mechanism and repository. This has brought them into a strong position within the content management system field. Moreover a healthy community is providing a lot of extensions as well as a lot of documentation for the system.

3.4 osCommerce

osCommerce [osCo08c] is one of the most wide spread open source eCommerce [Wiki] systems. It is developed with PHP and a flexible database connection in the background. It has proved itself as a successful project in the last years with more than 13,000 [osCo08b] registered shops and more than 4,500 [osCo08a] community developed extensions.

Nevertheless, the software is the only example included to demonstrate an approach with some weak characteristics. It does not provide well encapsulated extension points for all possible extensions. Only for a limited set of defined modules e.g. for the payment options encapsulated extensions can be placed in the system. All other extensions are shipped as code fragments which have to be integrated manually into the core source code. This leads to a messy system hard to update, maintain and error-prone because of the complex integration of extensions. Also the interoperability of extensions can become a challenge if there are dependencies to the same core source code. The extension life cycle is also not supported. Removing extensions from the source code can only be done manually. Furthermore it is nearly impossible if the code changes have not been marked during the installation.

The only structure provided by the system is the directory structure shown in Figure 15 that can be used to separate business code from configurations (includes/configure.php and admin/includes/configure.php) and language packs (includes/languages/ and admin/includes/languages/).

The extension repository of osCommerce [osCo08a] consists of a web interface with upload and download capabilities. This represents a very low barrier to contribute an own extension. On the other hand there is nearly no defined structure or quality assurance for extensions except for user feedback about specific extensions. Since end of 2007 the project started to refactor the repository [Hara07]. In the future they will provide extended documentation support and capabilities for community members to clean up extensions not owned by them.

Characteristics like the defined system access, security or loading strategies are not handled because extensions can ship code for every part of the system.

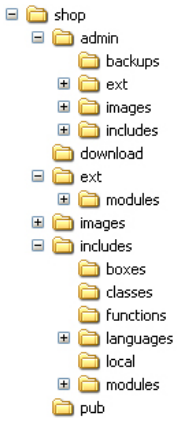


Figure 15. osCommerce Directory Structure

The same gets important for system evolutions. An update of osCommerce itself is like building up a complete new system if extensions have been implemented in the core code.

osCommerce is one of the projects with a self grown and not planed extension mechanism. In the authors opinion, this project was able to become one of the market leaders but will get under pressure in the next years, not at least because of its weak extension mechanism. There are upcoming products taking care of extension mechanisms from the beginning (e.g. Magento Commerce [Irub08])

4 Designing Extensible Software

While it makes sense to introduce an extension mechanism either to build an open system or just to support the internal development this has to be done in the right way.

The first step is to get aware and sensible of the characteristics. It does not make sense to introduce an extension mechanism just because of any hype. The demands for a good extension mechanism are too high.

For a good design of the extension mechanism the requirements of the software, the target groups and the system environment should be clarified. This is necessary for the design decisions later on.

The characteristics in this document can be used as a checklist when designing the extension mechanism.

An important point is to separate between a completely new development and the introduction into an existing product. Starting from scratch is the easiest way as in most cases. In the second case, a lot more constraints have to be considered. With existing software, the system access characteristic might be the first one should take care of. Introducing an extension mechanism right into the core might be possible only if the software already has a good architecture

as it was the case for the Eclipse system and Equinox. Otherwise one has the possibility of building up a second core and to migrate the system step by step. So defining the scope of the extension mechanism is a key factor for a successful extension mechanism.

If this is done, the following steps are nearly the same for existing and for completely new systems. It is important to create new extension points with as much care as possible but as flexible as necessary and to introduce only those extensions which are required. An extension point which is not used may be dead forever but increase complexity and costs.

Once one has considered all characteristics and designed the system you are ready to start the implementation can begin. If an extension mechanism is not only designed for UI purposes a special focus on API design is required. This is one of the most important and durable parts and architects without experience in this field should consider some literature or get some training before. The implement of the extension mechanism should be done in parallel with some example extensions to ensure the usability and to show how to use the mechanism. Test-extensions for the interfaces are also very useful tools for the development itself. This supports the documentation later on as well as frequent and automatic tests.

As with other new technics, an essential recommendation is to start small and to grow with the gathered experience.

Another aspect you should worry about is the fact that most of the effort in the software life cycle is spent on maintenance. This rule does not change for extension mechanisms. A good mechanism can support the maintenance of the software itself but also the extensions and the extension mechanism itself has to be maintained and will evolve in the future. So this has to be kept in mind.

As with most other technologies, it is important to take a look on existing systems to learn from their good and bad work. The examples provided in this paper already introduced a wide range of possibilities but taking a closer look to other systems will gather a lot of additional experience.

5 Conclusion

Most of the todays software architects have already developed an extensible software or at least have been in contact with one. Some of them may have taken care for issues covered by the related work but only a few have explicitly talked about all characteristics of extension mechanisms. This paper presented a list of characteristics which are important for this.

Extension mechanisms can support the general requirements of faster, cheaper, more flexible and evolving software development. Those goals are mainly achieved by the support of distributed development, breaking down complexity, reusability and more controlled software evolution. With a good extension mechanism it is also possible to deliver early releases with a subset of functionality and to develop product families.

Extension mechanisms can lead to a better software but only if they are done right. On the other side a bad extension mechanism can result in more complexity, less efficiency and less acceptance by the developers and users.

Extension mechanisms are once more not the silver bullet but even if the challenges of software development are not solved they are supported in a very advanced way. At least their concepts should be known and used by every software architect.

6 Appendix

References

AlLC07. Alexandre Alvaro, Rikard Land und Ivica Crnkovic. Software Component Evaluation: A Theoretical Study on Component Selection and Certification. *MRCT Report*, 2007.

Bloc06. Joshua Bloch. How to design a good API and why it matters. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, New York, NY, USA, 2006. ACM, S. 506–507.

CzEi02. Krzysztof Czarnecki und Ulrich Eisenecker. *Generative programming.* Addison Wesley. 2002.

Ecli08a. Eclipse Foundation. Rich Client Platform - Eclipsepedia, 2008. Available at: `http://wiki.eclipse.org/index.php/Rich_Client_Platform`; Last retrieved 2008-05-11.

Ecli08b. Eclipse Plugin Central. Eclipse Plugin Central, 2008. Available at: `http://www.eclipseplugincentral.com/`; Last retrieved 2008-06-07.

Equi08. Equinox Commiters. Equinox, 2008. Available at: `http://www.eclipse.org/equinox/`; Last retrieved 2008-06-07.

GrSh04. Jack Greenfield und Keith Short. *Software factories.* Wiley. 2004.

Hara07. Harald Ponce de Leon. osCommerce Blog ■ Blog Archive ■ New Community Add-Ons Site:, 2007. Available at: `http://blogs.oscommerce.com/2007/09/22/new-community-add-ons-site/`; Last retrieved 2008-05-28.

Irub08. Irubin Consulting Inc. DBA Varien. Magento - Home - Open Source eCommerce evolved, 2008. Available at: `http://www.magentocommerce.com/`; Last retrieved 2008-06-08.

osCo08a. osCommerce. osCommerce - Community Add Ons, 2008. Available at: `http://addons.oscommerce.com/`; Last retrieved 2008-05-28.

osCo08b. osCommerce. osCommerce - Live Shops Directory, 2008. Available at: `http://shops.oscommerce.com/`; Last retrieved 2008-05-28.

osCo08c. osCommerce. osCommerce - Open Source E-Commerce Solutions, 2008. Available at: `http://oscommerce.org/`; Last retrieved 2008-05-24.

OSGi08a. OSGi Alliance. OSGi Alliance | About / OSGi Technology, 2008. Available at: `http://www.osgi.org/About/Technology`; Last retrieved 2008-06-30.

OSGi08b. OSGi Alliance. OSGi Alliance | Main / OSGi Alliance, 2008. Available at: `http://www.osgi.org`; Last retrieved 2008-05-11.

Parn78. David Lorge Parnas. Designing Software for Ease of Extension and Contraction. In *Proceedings of the Third International Conference on Software Engineering*, 10-12 1978, S. 264–277.

Skaa07. Kasper Skaarhoj. *Typo3 Core Documentation.* Typo3, April 2007. Available at: `http://typo3.org/documentation/document-library/core-documentation/doc_core_inside/current/view/2/1/`; Last retrieved 2008-06-24.

SzGM02. Clemens Szyperski, Dominik Gruntz und Stephan Murer. *Component Software: Beyond Object-Oriented Programming.* ACM Press and Addison-Wesley, New York, NY. 2002.

Typo08a. Typo3 Association. Typo3 Content Management System, 2008. Available at: http://www.typo3.org; Last retrieved 2008-05-11.

Typo08b. Typo3 Association. Typo3 Extension Keys, 2008. Available at: http://typo3.org/extensions/extension-keys/; Last retrieved 2008-05-28.

Typo08c. Typo3 Security Team. Typo3 What are reviews, 2008. Available at: http://typo3.org/extensions/what-are-reviews/; Last retrieved 2008-05-28.

Wiki. Wikipedia. eCommerce. Available at: http://en.wikipedia.org/wiki/Electronic_commerce; Last retrieved 2008-05-18.

Graphical Editing Framework

Maxim Kremer

Betreuer: Dipl.-Inform. Johannes Stammel

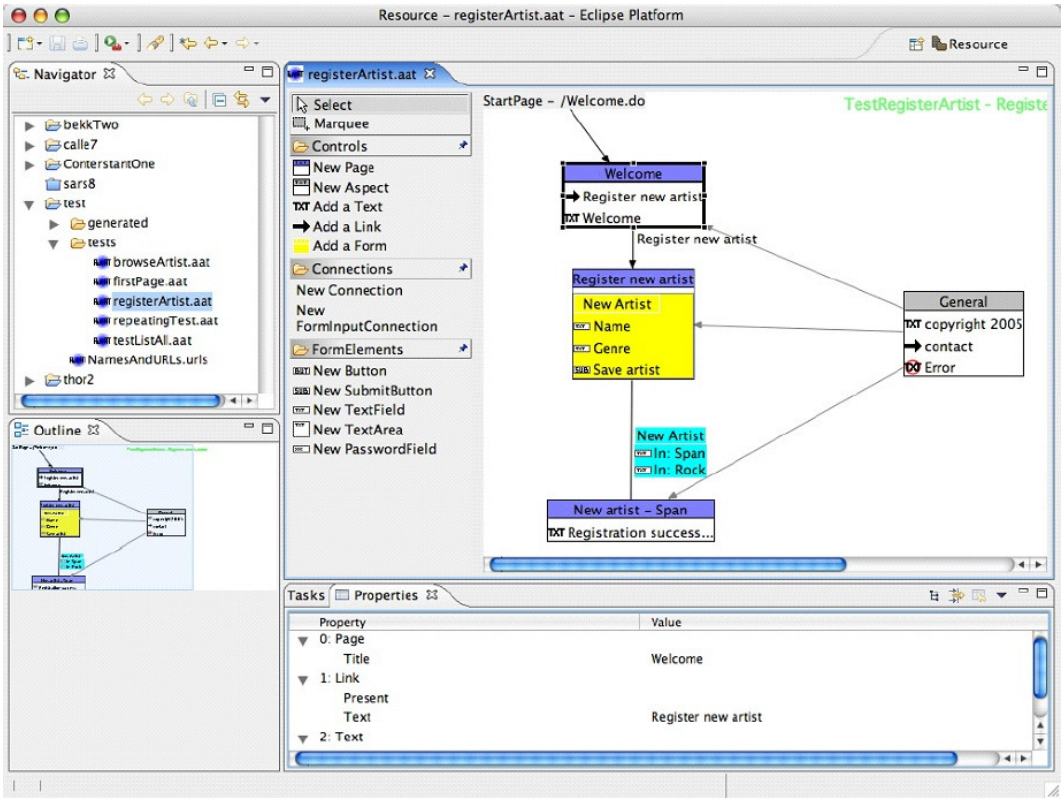
Zusammenfassung Die Ausarbeitung stellt eine Einführung in das Graphical Editing Framework(GEF) dar. Zuerst werden Aufgaben und Anwendungsgebiete des GEF erklärt und am Beispiel eines GEF-basierten Editors veranschaulicht. Dann wird es auf die innere Architektur des Frameworks eingegangen. Anschließend werden Vor- und Nachteile von GEF erläutert und Alternativen vorgestellt.

1 Einführung und Motivation

1.1 Aufgaben des Graphical Editing Frameworks

Das Graphical Editing Framework(GEF) ist ein Open Source Plugin, das einen Programmrahmen bereitstellt, mit dem sich für nahezu jedes beliebige existierende Modell ein graphisches Editor-Plugin erstellen lässt. Der Editor kann dem Benutzer gebräuchliche Funktionen wie drag and drop, copy und paste sowie Menüs und Werkzeugleisten zur Verfügung stellen und somit lässt sich das Modell auf verschiedenste Arten einfach und übersichtlich verändern. [Bill04, S.88]

Screenshot aus AutAT, einem GEF-basierten Eclipse Plugin zum Testen von Web-Anwendungen:



[aut, S.1]

GEF wird oft in Verbindung mit EMF(Eclipse Modelling Framework) genutzt. EMF ist ein Java-Framework, das aus einem Modell automatisch Java-Quellcode erzeugen und somit dem Entwickler viel Zeit sparen kann. Mit EMF lassen sich Modellinstanzen schnell erstellen, manipulieren, abfragen und auf Änderungen überwachen. Anschließend kann man den von EMF generierten Modellcode mit Hilfe von GEF in das graphische Editor-Plugin integrieren. Die zusätzliche Benutzung von EMF kann also die Entwicklung eines GEF-basierten Editor-Plugins erheblich vereinfachen.

1.2 Einsatzgebiete

Das Graphical Editing Framework ist ein mächtiges Werkzeug, das sehr breit eingesetzt werden kann. Ursprünglich wurde es entwickelt, um Modellierungswerkzeuge zu entwerfen. So können auf der Basis von GEF diverse UML-Editoren für Zustands-, Aktivitäts- oder Klassendiagramme implementiert werden. Seit neuster Zeit wird GEF auch dazu verwendet, visuelle GUI-Editoren zu entwickeln.

So ist zum Beispiel der Eclipse Visual Editor eine GEF-basierte Anwendung. [Daum05, S.349]

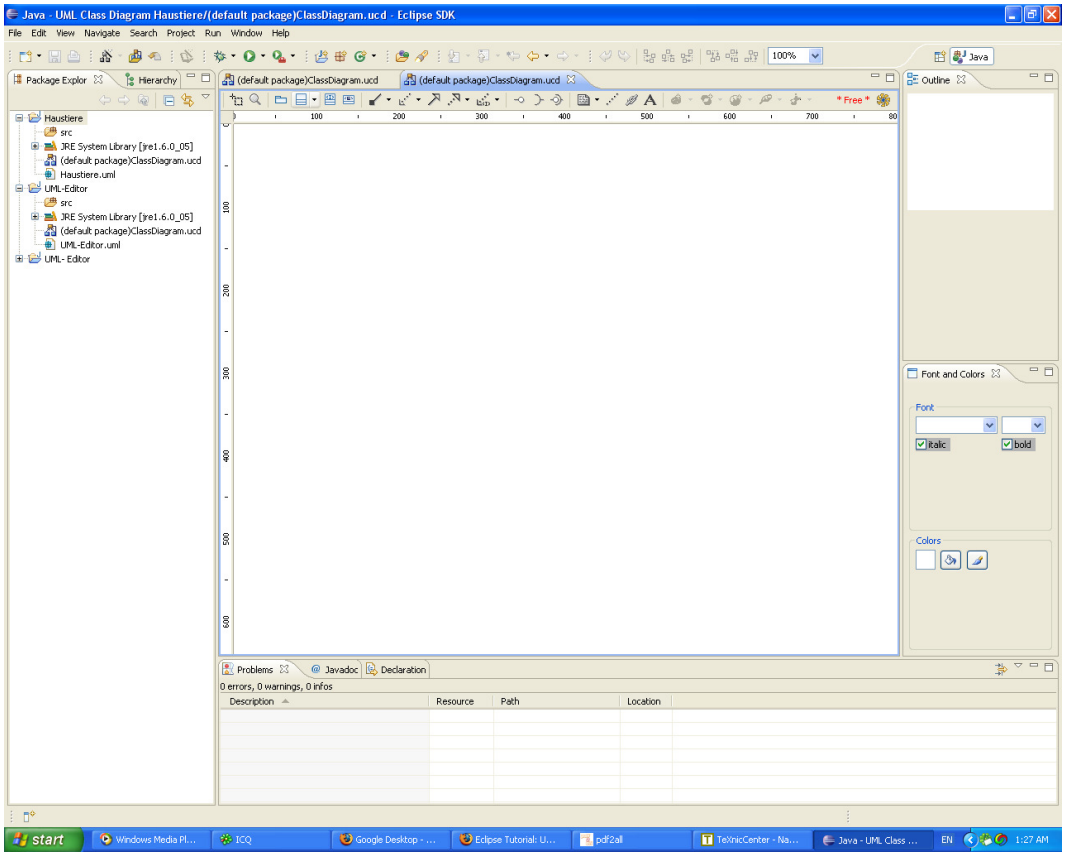
2 Beispiel: OMONDO - ein GEF-basierter UML-Editor

Die Entwicklung eines Editors mit Hilfe von GEF ist arbeitsaufwändig und erfordert fundierte Vorkenntnisse in Plugin-Entwicklung für Eclipse. Eine ausführliche Vorstellung eines solchen Prozesses würde den Rahmen einer Proseminar-Ausarbeitung sprengen. Stattdessen wird im Folgenden ein graphisches Eclipse Plugin vorgestellt, das auf GEF und EMF basiert.

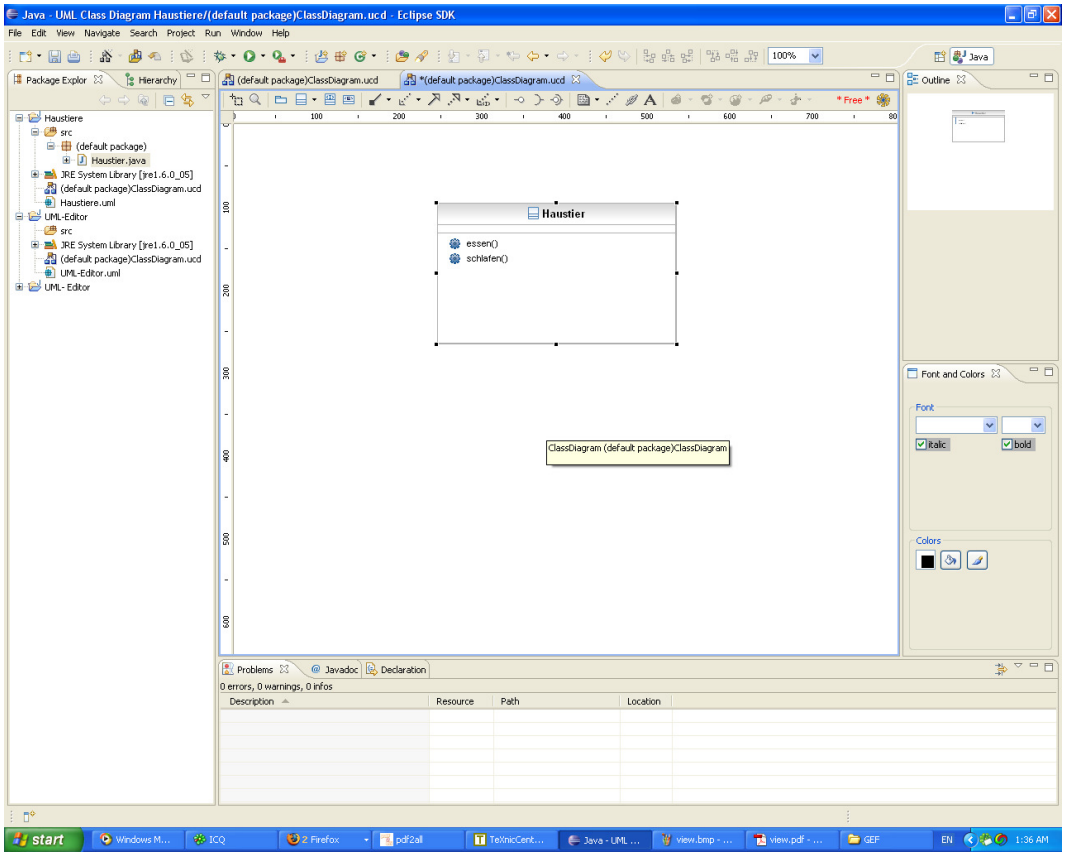
Dieses kleine Beispiel soll die Funktionen eines typischen GEF-basierten Editors demonstrieren und verdeutlichen, für welche Art von Anwendungen sich GEF als Entwicklungsbasis eignet.

OMONDO ist ein kommerzieller, in der Grundversion kostenloser UML-Editor. Der Benutzer kann UML-Diagramme erstellen und modifizieren, die von OMONDO direkt in Java-Code umgesetzt werden. Automatisches Übertragen der Änderungen des Codes durch den Benutzer in das entsprechende UML-Diagramm wird von OMONDO ebenfalls unterstützt.

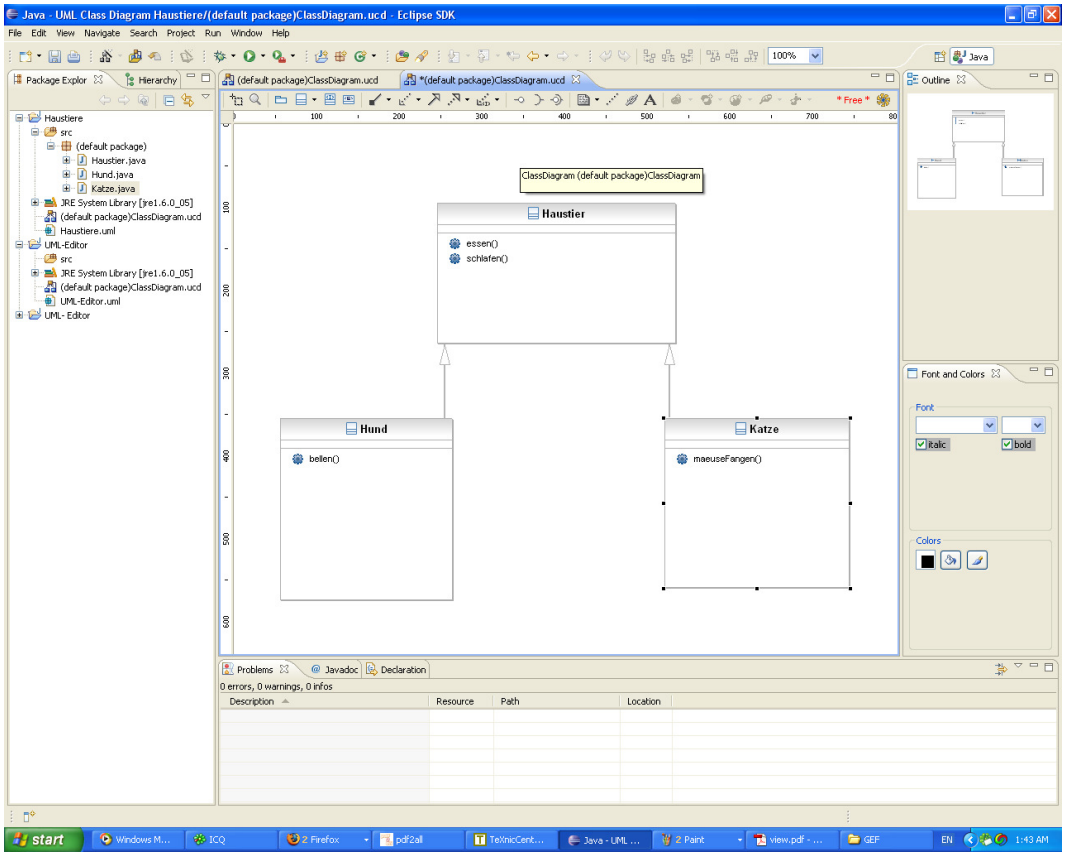
Nach der Installation öffnen wir Eclipse und erstellen ein neues Java-Projekt, das wir „Haustiere“ nennen. Danach klicken wir mit der rechten Maustaste auf das erzeugte Paket und gehen auf „New -> Other -> UML Diagramms -> UML Class Diagramm“.



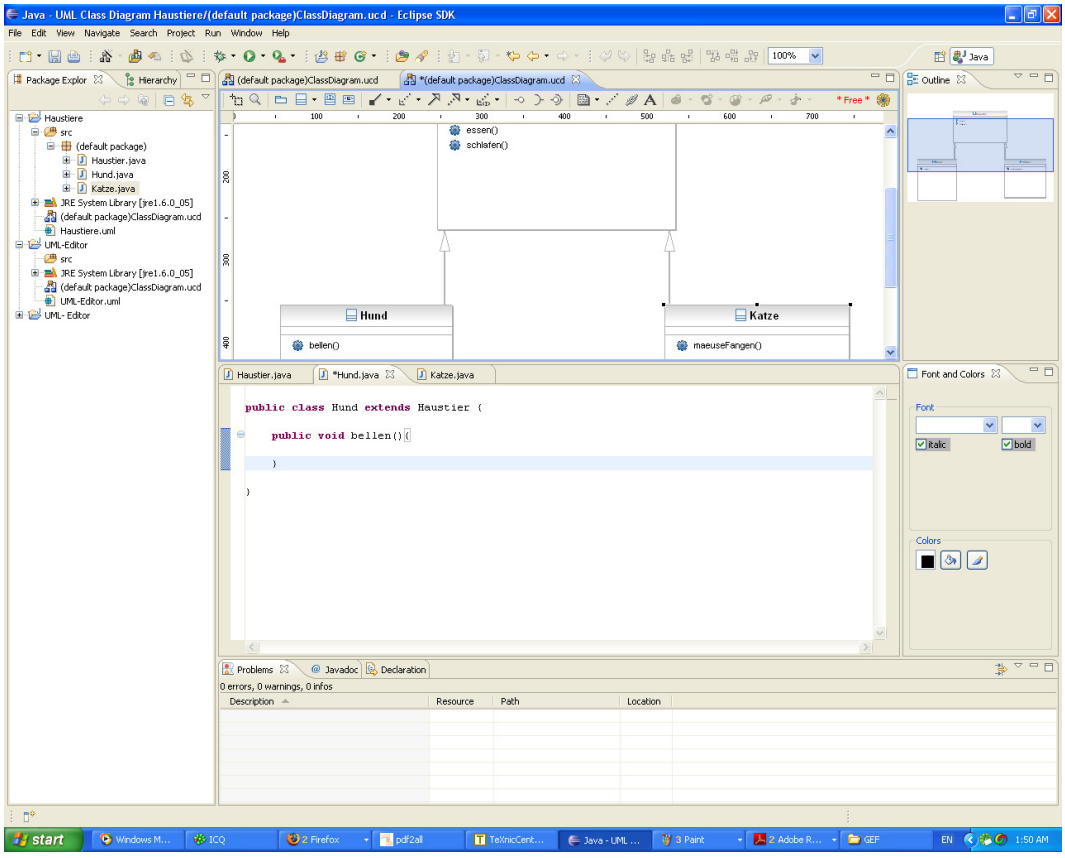
Es erscheint eine neue Ansicht, in der man direkt das UML-Klassendiagramm zeichnen kann. Oben sieht man eine Leiste mit den Werkzeugen zur Darstellung der Elementen eines Klassendiagramms. Mit dem entsprechenden Werkzeug erzeugen wir eine neue Klasse. Hier können wir festlegen, wie groß das Kästchen sein soll, das die Klasse repräsentiert und wo es im UML-Diagramm erscheinen soll. Es öffnet sich ein Fenster, in dem man gleich einige Eigenschaften der neuen Klasse festlegen kann. Wir nennen sie „Haustier“. Nun erscheint die neue Klasse in der Übersicht in dem linken Teil des Bildschirms. Klickt man nun mit der rechten Maustaste auf die Klasse, so kann man der Klasse neue Methoden und Attribute hinzufügen. Wir fügen unserer Klasse die void-Methoden essen() und schlafen() hinzu.



Analog erstellen wir 2 weitere Klassen und plazieren diese in dem UML-Diagramm unter der „Haustier“-Klasse. Wir nennen die neuen Klassen „Hund“ und „Katze“ und versehen diese mit den mit den Methoden bellen() bzw maeuse-Fangen(). Nun wollen wir zeigen, dass die zwei Klassen von der Klasse „Haustier“ abstammen und somit auch über die Methoden „essen“ und „schlafen“ verfügen. Die Vererbung wird bei UML durch einen Pfeil mit leerer Spitze dargestellt, der von der abstammenden auf die übergeordnete Klasse zeigt. Wir wählen das entsprechende Werkzeug und zeichnen die Vererbungspfeile in das UML-Diagramm ein. Alle bis jetzt erzeugten Klassen können in Übersicht eingesehen werden.



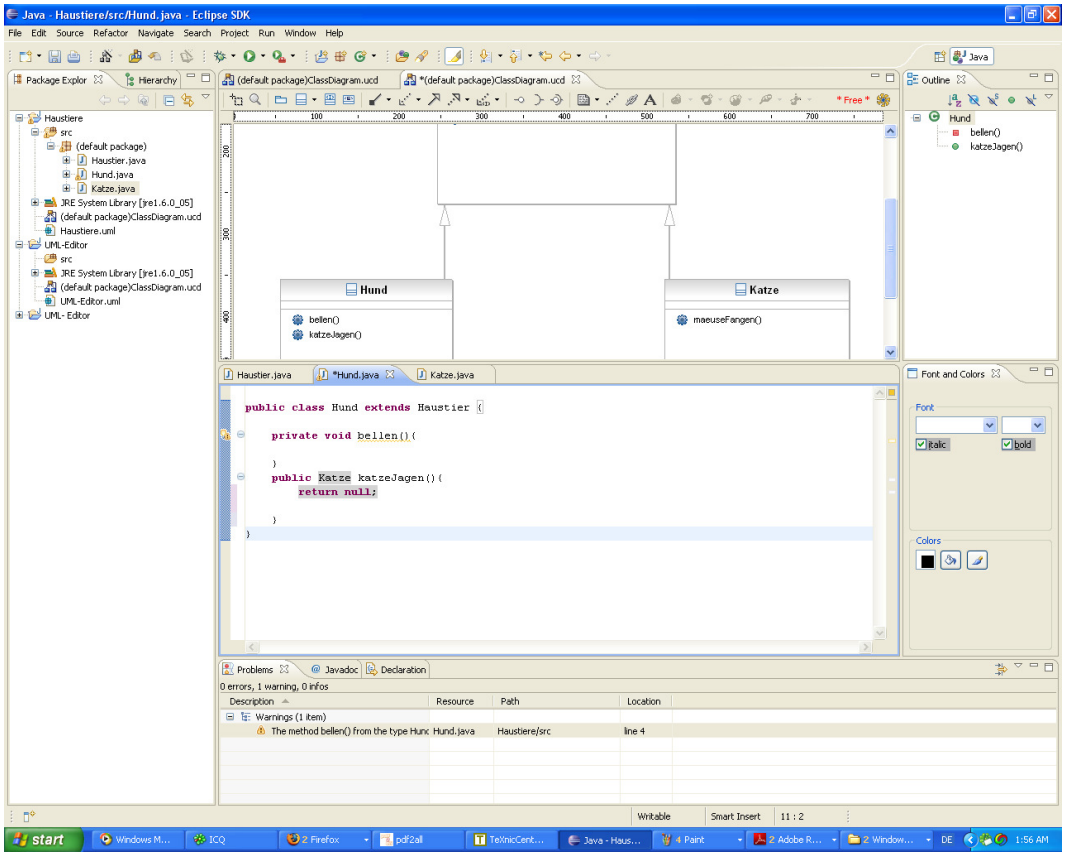
Nun machen wir einen Doppelklick z.B. auf die Klasse „Hund“. Es öffnet sich im unteren Teil des Bildschirms ein Fenster, in dem wir den entstandenen Java-Code sehen. Dieser enthält alle Informationen die im UML-Diagramm enthalten sind: Die Klasse „Hund“ ist eine öffentliche Klasse, die von der Klasse „Haustier“ abstammt. Darauf weist im Code das Schlüsselwort „extends“ hin. Ausserdem verfügt die Klasse über die Methode „bellen“ ohne Rückgabewert. Im gleichen Fenster kann der Entwickler nun die Methode implementieren oder die Klassenstruktur verändern. Alle Änderungen werden automatisch in das UML-Diagramm übertragen.



Nun können wir zum Beispiel im Code der Klasse „Hund“ die Methode katzeJagen hinzufügen. Diesmal soll es keine void-Methode sein sondern einen Rückgabewert vom Typ „Katze“ liefern. Wir schreiben diese neue Methode in unseren Code rein. Dazu müssen wir nur folgende Zeile hinzufügen

public Katze katzeJagen

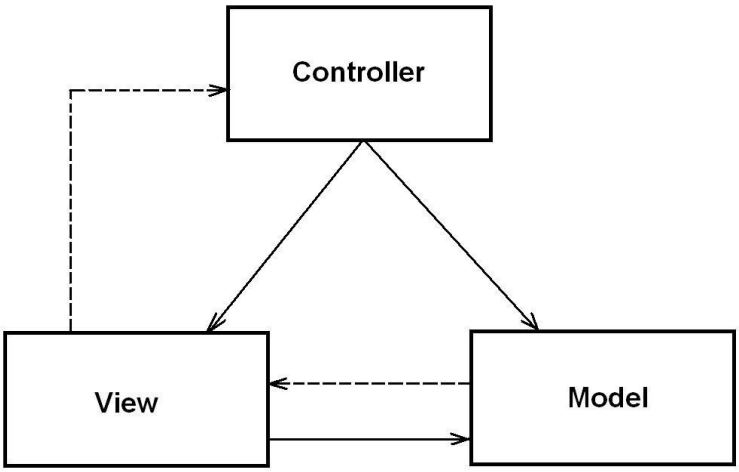
und mit geschweiften Klammern den Platz für die Implementierung markieren. Im oberen Fenster mit der grafischen Darstellung können wir sehen, dass unser UML-Diagramm von OMONDO sofort aktualisiert wird. Im Kästchen „Hund“ steht nun auch die neue Methode „katzejagen“.



3 Grundelemente und Innere Architektur

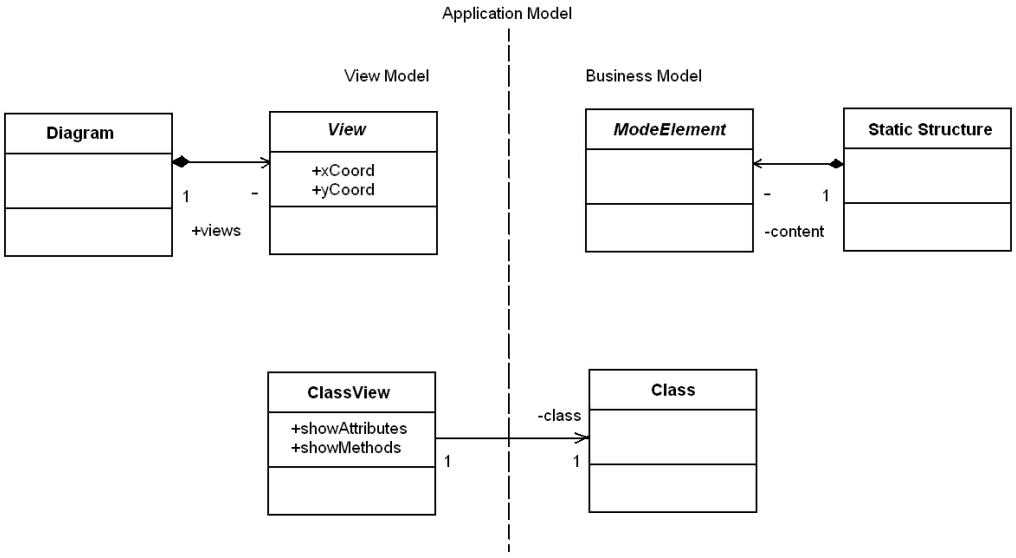
3.1 Die Model-View-Controller(MVC) - Architektur

GEF basiert auf der Model-View-Controller Architektur d.h. die Software-Entwicklung ist in 3 Einheiten strukturiert: Datenmodell, Präsentation und Programmsteuerung. Dadurch lassen sich einzelne Komponenten wiederverwenden und eine weitere Erweiterung oder Änderung werden erleichtert. Die folgende Grafik veranschaulicht das Prinzip der MVC-Architektur



[wiki08, S.1]:

Das Modell enthält die darzustellenden Daten und gegebenenfalls die Logik des Softwaresystems. Es ist von Präsentation und Programmsteuerung unabhängig. Man unterscheidet zwischen den sogenannten „Business-Model“ und „View-Model“. Das erste enthählt nur wichtige semantische Details, während die präsentationsspezifischen Daten wie zum Beispiel die x-y-Koordinaten, an denen die Klasse in der Ansicht erscheint, im View-Model gespeichert sind.



[Huds03, S.1]

Die Präsentation stellt die relevanten Daten aus dem Modell dar und nimmt Benutzerinteraktionen entgegen. Sie verfügt sowohl über Informationen zu ihrer Steuerung als auch über die Daten aus dem Modell. Die Weiterverarbeitung der vom Benutzer übergebenen Daten ist allerdings nicht die Aufgabe der Präsentation.

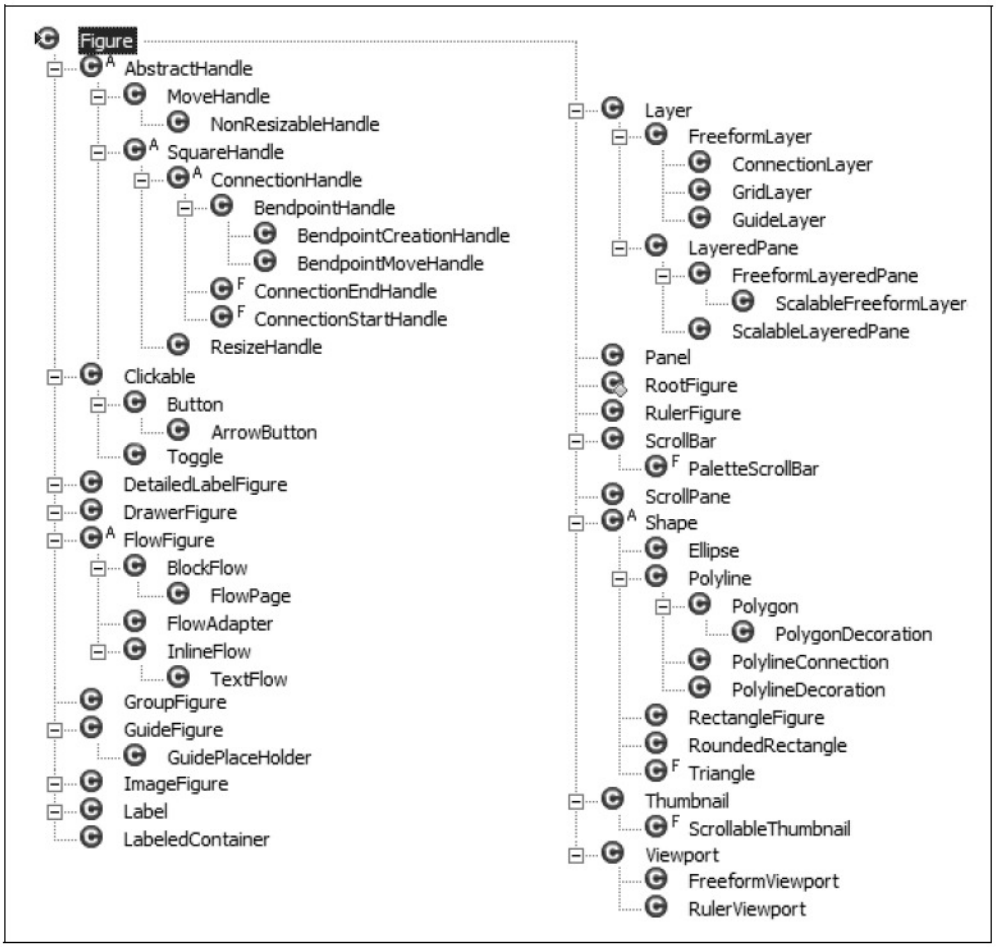
Die Steuerung verwaltet die Präsentationen, nimmt von ihnen Benutzeraktionen entgegen, wertet diese aus und entscheidet daraufhin, welche Daten im Modell geändert werden müssen. Ausserdem ist es die Aufgabe der Steuerung, die Benutzerinteraktionen in der Präsentation einzuschränken. [Huds03, S.1]

3.2 Figures

Im nächten Schritt muss der Entwickler entscheiden, wie das Modell dargestellt werden soll. Dazu werden die graphischen Objekte des Draw-2D Plugins verwendet. Diese stammen von der Basisklasse „Figure“ ab, die die gemeinsamen Eigenschaften aller graphischen Objekte definiert.

Alle Maus- und Tastaturereignisse sind als Attribute der jeweiligen Figures definiert und müssen vom Entwickler implementiert werden. Einige Figuren können direkt verwendet werden um Objekte aus dem Modell darzustellen. Manchmal muss man auch mehrere Figuren und Layoutmanager kombinieren. Der Entwickler kann auch eigene Figuren implementieren und deren Aussehen selbst definieren. Die meisten Probleme lassen sich jedoch mithilfe der mitgelieferten Layoutmanager und Figuren lösen.

Die folgende Graphik ist eine Übersicht über die konkreten und abstrakten Unterklassen der Klasse „Figure“:



[Daum05, S.351, Abb 16-1]

3.3 EditParts

Die Verbindung zwischen dem Modell und der Präsentation wird durch die „controllers“ (Programmsteuerung) umgesetzt. Bei GEF sind es die sogenannten EditParts.

EditParts sind die zentralen Elemente einer GEF-Anwendung. Jedes Objekt aus dem Modell, das graphisch dargestellt werden soll, ist mit seier Visualisierung in der Präsentation duch einen EditPart verbunden. Dieser erhält Daten aus dem Modellobjekt. Sobald die Daten geändert werden aktualisiert der EditPart die graphische Darstellung in der Präsentation.

Mit Hilfe der EditParts werden also die graphischen Objekte realisiert. Die Struktur eines EditParts hängt von dem Objekttyp ab, der duch ihn repräsentiert werden soll. Eine spezielle EditPartFactory-Instanz erzeugt eine EditPart-Instanz, die zu dem Objekt, das abgebildet werden soll, passt.

Ein Modell wird mit einem Graphical Viewer angezeigt, der es über die Methode setContents() bekommt. Davor muss für den Viewer mit der Methode setEditPartfactory() erst eine EditPartFactory registriert werden, die zu jedem Modellobjekt einen entsprechenden GraphicalEditPart erzeugt. [Daum05, S.354]

Man unterscheidet 3 Typen von EditParts.

Graphische EditParts (Graphical EditParts) implementieren das Verhalten der graphischen Objekte.

VerbindungsEditParts (ConnectionEditParts) legen den Zusammenhang zwischen graphischen EditParts fest.

Das dritte Typ TreeEditParts wird für die Darstellung des Modells als Baumstruktur verwendet und ist für die Erstellung graphischer Editoren eher irrelevant. [Bill04, S.104]

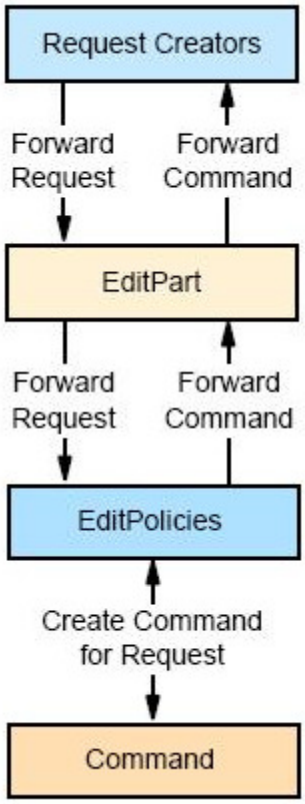
Die Konstruktion von EditParts ist parallel zur Entstehung der Ansicht, d.h. ein EditPart wird erst aktiv, wenn das entsprechende Modellobjekt vom Benutzer hinzugefügt wird und das GEF darüber informiert wird. Wird ein EditPart nicht mehr gebraucht, so wird er deaktiviert. Dadurch werden Systemressourcen gespart. Man spricht vom Lebenszyklus eines EditParts.

3.4 Kommandos

Die eigentlichen Veränderungen am Modell erfolgen über Kommandos. Der Entwickler muss auf der Grundlage einer von GEF bereitgestellten abstrakten Klasse „Command“ für alle möglichen Modifikationen des Modells durch den Benutzer(z.B Element löschen, neues Element hinzufügen) entsprechende Kommandos implementieren. Unter anderem muss für jede Kommandoklasse die „Undo“-Funktion realisiert werden.

3.5 Requests

Requests sind Objekte, die mit EditParts kommunizieren. Sie enthalten Informationen, die der EditPart benötigt, um diverse Funktionen auszuführen. Das folgende Diagramm beschreibt die Interaktionen zwischen einem Request und weiteren beteiligten Objekten:

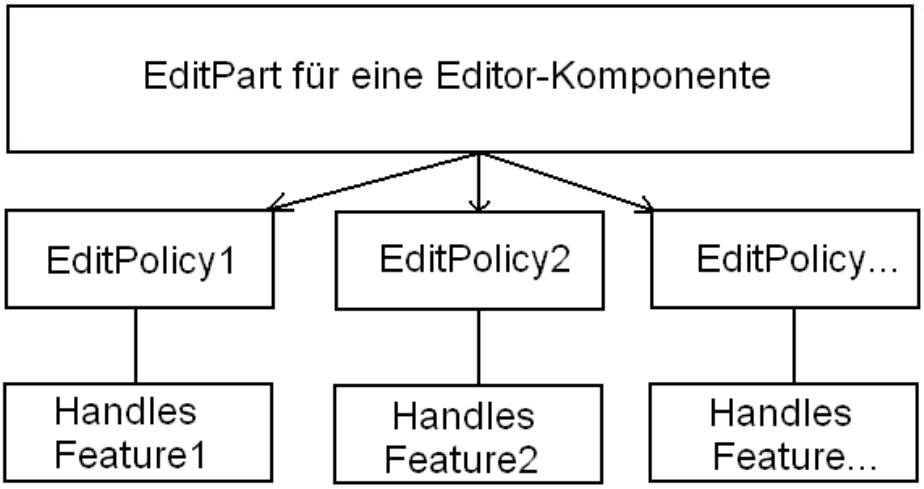


[Bill04, S.106, Abb 3-9]

Nachdem der Request erstellt wurde, wird er an den EditPart weitergeleitet. Anstatt den Request selbst zu bearbeiten leitet ihn der EditPart an eine seiner EditPolicies (werden im nächsten Unterkapitel behandelt). Diese kann den Request interpretieren und erstellt eine entsprechende Kommando-Instanz, die anschließend ausgeführt wird, um den Request zu erfüllen. [Bill04, S.106] Je nach Art des Requests wird eine der mehreren EditPolicies des betroffenen EditParts aufgerufen.

3.6 EditPolicies

Wie sich Benutzerinteraktionen in der Präsentation auf das Modell auswirken wird in den sogenannten EditPolicies festgelegt. Sie stellen die Schnittstelle zum Modell bereit und werden vom EditPart des entsprechenden Modellobjekts verwaltet. Die EditPolicies sind somit für die Editor-Funktionalität der EditParts zuständig. Ein EditPart stellt mehrere EditPolicies zur Verfügung - eine für jede Art der Benutzeriteraktion mit dem korrespondierendem Objekt.



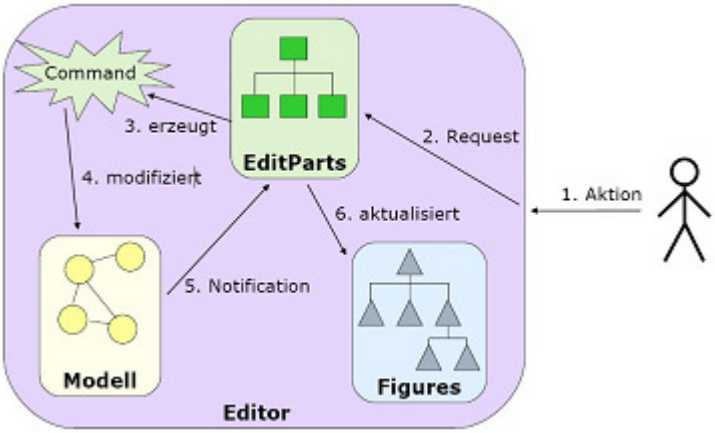
[Gabr, S.9]

Die verschiedenen EditPolicies werden nach ihren „Rollen“ kategorosiert.

- ComponentEditPolicy behandelt Requests, die nichts mit dem UI zu tun haben, sondern sich nur auf ein Modellobjekt beziehen (z.B. Objekt löschen).
- ConnectionEditPolicy werden von Connection EditParts benutzt.
- ContainerEditPolicy übernimmt Container-Operationen (z.B. Erzeugung eines Kind-EditParts)
- LayoutEditPolicy übernimmt layoutgebundene Container-Operationen (z.B. wo werden die erzeugten Kind-EditParts plaziert).
- GraphicalNodeEditPolicy erzeugt und verwaltet Verbindungen zwischen den EditParts.
- Mit Hilfe von DirectEditPolicy kann der Benutzer durch einen Doppelklick auf einer Figur die Eigenschaften des entsprechenden Objekts direkt verändern. [Bill04, S.108]

Die Verwendung der Rollennamen ist lediglich eine Konvention, da der Entwickler das Verhalten verschiedener EditParts komplett selbst implementieren muss. [Zoio04]

Die Folgende Graphik veranschaulicht die innere Architektur des GEF und den Zusammenhang zwischen den Grundelementen:



[Joel, S.13]

4 Fazit und Ausblick

4.1 Vor- und Nachteile des GEF

Die Menge der Anwendungsfälle, für die sich GEF eignet ist begrenzt. Bei einigen Projekten lässt sich mit seiner Anwendung jedoch viel Zeit bei der Implementierung sparen. Das Framework bietet folgende Vorteile:

- Beliebiges Datenmodell kann visualisiert werden.
- Für das Modifizieren des Modells wird eine breite Werkzeugpalette bereitgestellt.
- Geläufige Funktionen wie drag-and-drop oder copy and paste werden unterstützt.
- Druckausgabe wird unterstützt.
- Viele vorgefertigte Funktionen werden bereitgestellt.

Da das GEF ein ziemlich junges Projekt ist, hat es auch einige Nachteile. Die drei wichtigsten sind:

- Mit GEF muss der Entwickler noch sehr viel manuell implementieren. Mit Hilfe von GMF, einem Erweiterungsplugin, das mit GEF und EMF arbeitet, lassen sich graphische Editoren automatisch erzeugen.
- Die unausführliche Dokumentation erschwert den Lernprozess.
- Viele Konzepte eignen sich nicht für ein breites Anwendungsspektrum.

[Daum05, S.350]

4.2 Alternativen

Eine direkte, in die Eclipse-Umgebung integrierte Alternative zu GEF gibt es nicht. Je nach Anwendungsfall bieten sich aber folgende leistungsstarke Möglichkeiten:

- Für die Erstellung graphischer Oberflächen mit Java eignet sich das Standard Widget Toolkit (SWT).
- Interaktive zweidimensionale Grafiken können unter Verwendung von SVG (Scalable Vector Graphics) in der XML-Syntax beschrieben werden. Die Grafiken können z.B. unter Verwendung von Apache Batik in eine Eclipse-Plattform eingebettet werden.

[Daum05, S.383]

5 Zusammenfassung

GEF ist ein Eclipse-Plugin mit dem sich graphische Editor-Plugins entwickeln lassen. Die Hauptanwendungsgebiete sind Modellierungswerkzeuge und GUI-Builder. Durch die zusätzliche Verwendung der Plugins EMF und GMF kann die Erstellung eines Editors vereinfacht werden.

Als Beispiel einer GEF-Anwendung haben wir das Plugin OMONDO behandelt. Mit dessen Hilfe lassen sich UML-Diagramme einfach erstellen. OMONDO erzeugt automatsch Java-Code zu einem UML-Diagramm und kann Änderungen im Code in das UML-Diagramm übertragen.

Danach wurde ein Einblick in die Innere Architektur von GEF gegeben. Alle Modellobjekte werden durch die graphischen Objekte des Draw2D-Plugins repräsentiert. Jedem Objekt entspricht ein grafischer EditPart, der die Verbindung zwischen dem Modell und der Präsentation umsetzt. Das Verhalten eines EditParts wird in seinen EditPolicies implementiert. Eine Benutzerinteraktion ruft einen Request hervor, der an die entsprechende EditPolicy weitergeleitet wird. Diese erzeugt ein Kommando, das anschließend ausgeführt wird, um den Request zu erfüllen.

GEF ist also ein mächtiges Werkzeug, mit dem sich für ein beliebiges Datenmodell ein Editor erstellen lässt, allerdings ist die Entwicklung zeit- und arbeitsaufwändig. Als Alternative bietet sich unter anderem das Standard Widget Toolkit an.

Literatur

aut. Informationsseite über das Plugin AutAt.

Bill04. Anna Gerber Gunnar Wagenknecht Philippe Vanderheyden Bill Moore, David Dean. Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, Feb 2004.

Daum05. Berthold Daum. Rich-Client-Entwicklung mit Eclipse 3.1, ISBN 3898643530, 2005.

Gabr. Karsten Ehrig Gabriele Taentzer. Grafische Editor Framework (GEF) von Eclipse Teil 2 - Powerpoint Präsentation.

Huds03. Randy Hudson. GEF-Tutorial auf der Offiziellen Eclipse-Seite, Jul 2003.

Joel. Dietrich Travkin Joel Greenyer. Einführung in Eclipse & GEF.

wiki08. Wikipedia-Eintrag „Model-View-Conbtroller“, Jul 2008.

Zoio04. Phil Zoio. Building a Database Schema Diagram Editor with GEF, Sep 2004.