universität wien

MASTERARBEIT

Titel der Masterarbeit

„Ein webbasierter kollaborativer C&C-Editor zur Dokumentierung der Softwarearchitektur“

Verfasser

Weiqi LI

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2015

Studienkennzahl lt. Studienblatt:	**A 066 926**
Studienrichtung lt. Studienblatt:	**Masterstudium Wirtschaftsinformatik**
Betreuer:	**Univ.-Prof. Dr. Uwe Zdun**

Danksagung

An dieser Stelle möchte ich mich bei all jenen Personen bedanken, die mir während der Erstellung meiner Masterarbeit geholfen haben:

- Frau Dipl. -Ing. Ioanna Lytra und Herren Univ. -Prof. Dr. Uwe Zdun für die angenehme Betreuung sowie die schnelle Rückmeldung und die gute Hilfestellung bei diverse Fragen im Zusammenhang mit dieser Arbeit.

- meinen Eltern und ganz speziell meiner Frau Liyan Hu, die mir mit guten Worten und im Gebet zur Seite gestanden sind und mir so die Kraft gegeben haben, diese Arbeit viel Elan und Einsatz zu schreiben.

Euch allen vielen Dank!

Weiqi Li Wien, im Januar 2015

Abstrakt

Für den Entwurf und die Implementierung eines komplexen Softwaresystems ist eine Softwarearchitektur notwendig. Um die Vorteile der Softwarearchitektur zu nutzen und um deren Kosten zu senken muss die Softwarearchitektur dokumentiert werden. Im Rahmen der vorliegenden Arbeit wird die graphische Dokumentation von Softwarearchitekturen betrachtet. Die wichtigen Begriffe dafür wie z.B. Softwarearchitektursichten, Architekturmodelle, Architekturstile sowie graphische Architektur-Tools sind die Schwerpunkte dieser Arbeit. Im praktischen Teil wird das WebVbMF (**Web-** and **V**iew-**b**ased **M**odeling **F**ramework), eine webbasierte kollaborative Anwendung zur Dokumentation von Softwarearchitekturen, zur Ergänzung der existierenden graphischen Architektur-Tools entwickelt. Dadurch sollen einerseits die Methode der graphischen Dokumentation der Softwarearchitektur überprüft werden, und andererseits die Möglichkeiten, die Softwarearchitekturen kollaborativ zu dokumentieren, untersucht werden.

Inhaltsverzeichnis

1 Einleitung 1
1.1 Motivation 1
1.2 Zielsetzung 2
1.3 Aufbau der Arbeit 2
2 Softwarearchitektur 5
2.1 Geschichte 5
2.2 Begriff 7
2.3 Bedeutung 10
3 Dokumentation der Softwarearchitektur 17
3.1 Formen 17
3.2 Sichten 19
3.3 Architekturmodelle 22
3.3.1 4+1 Sichtenmodell 23
3.3.2 Garland And Anthony's Viewpoints 26
3.3.3 Rozanski And Woods's Model 29
3.4 Architekturstile 31
3.4.1 Wichtige Modulstile 32
3.4.2 Wichtige C&C-Stile 37
3.4.3 Wichtige Allokationsstile 42
3.5 Weg zur graphischen Dokumentation 44
4 Graphische Architektur-Tools 49
4.1 UML-basierte Anwendungen 49
4.2 yEd Graph Editor 53
4.3 VbMF 55
4.4 WebVbMF 57
5 WebVbMF Implementierung 61
5.1 Anforderungen und Herausforderungen 61

5.2 Webbasiertheit 63
5.3 Editor 70
5.4 Dateioperationen 79
5.5 Kollaboration 85
5.5.1 Sharing-Mechanismus 86
5.5.2 Kollaboration-Mechanismus 86
5.5.2.1 Asynchrone Kollaboration 88
5.5.2.2 Synchrone Kollaboration 89
5.5.3 Verlaufssicht 96
5.6 Persistenz 98
5.7 Zugriffskontrolle 101
6 Fazit 105
Anhang 107
Bedienungsanleitung 107
Literaturverzeichnis 113

Tabellenverzeichnis

Tabelle 1: Die wichtigen Modulstile 32

Tabelle 2: Die wichtigen C&C-Stile 37

Tabelle 3: Die wichtigen Allokationsstile 42

Tabelle 4: Die Stakeholder und ihr Interesse an den verschiedenen Sichten 45

Tabelle 5: Eine mögliche ACM im WebVbMF 102

Tabelle 6: RBAC-Rollen bei WebVbMF 104

Tabelle 7: Die Funktionen von WebVbMF 108

Tabelle 8: Die Funktionen des Editors 110

Abbildungsverzeichnis

Abbildung 1: Dokumentation der Softwarearchitektur als Bild 18
Abbildung 2: Einige Subsysteme des menschlichen Körpers 20
Abbildung 3: Die Klassifikation von architektonischen Strukturen und Sichten 21
Abbildung 4: Das 4+1 Sichtenmodell 23
Abbildung 5: Das Szenario eines CRM Systems 25
Abbildung 6: Ein Beispiel für den Logical View 25
Abbildung 7: Ein Beispiel für den Development View 26
Abbildung 8: Ein Beispiel für den Physical View 26
Abbildung 9: Die Sichten von Rozanski And Woods's Model 29
Abbildung 10: Ein Beispiel für den Zersetzungsstil 33
Abbildung 11: Ein Beispiel für den Gebrauchsstil 33
Abbildung 12: Ein Beispiel für den Verallgemeinerungsstil 34
Abbildung 13: Ein Beispiel für den Schichtenstil 34
Abbildung 14: Ein Beispiel für den Aspekte-Stil 35
Abbildung 15: Ein Beispiel für das Datenmodell 36
Abbildung 16: Ein Beispiel für den „Pipe-And-Filter“-Stil 38
Abbildung 17: Ein Beispiel für den Client-Server-Stil 38
Abbildung 18: Ein Beispiel für den „Peer-To-Peer“-Stil 39
Abbildung 19: Ein Beispiel für den SOA-Stil 40
Abbildung 20: Ein Beispiel für den „Shared-Data“-Stil 41
Abbildung 21: Ein Beispiel für den Deployment-Stil 43
Abbildung 22: Ein Beispiel für den Installationsstil 44
Abbildung 23: Die Templates im RSA 51
Abbildung 24: Die Topologie einer Anwendung 52
Abbildung 25: Ein Sketch 52
Abbildung 26: Die möglichen Elementtypen in dem yEd Graph Editor 53
Abbildung 27: Das mit dem yEd Graph Editor erzeugte Anwendungsfalldiagramm 54

Abbildung 28: Mit dem yEd Graph Editor erzeugtes nicht standardisiertes Diagramm 55
Abbildung 29: Die Diagrammarten im VbMF 56
Abbildung 30: Die View-Formen im VbMF 56
Abbildung 31: Das Komponentendiagramm des Adventure Builders 57
Abbildung 32: Das C&C-Diagramm von Adventure Builder 58
Abbildung 33: Mit AcmeStudio dokumentierte Struktur 59
Abbildung 34: Das C&C-Diagramm des in der Abbildung 33 gezeigten Diagramms 59
Abbildung 35: Die Registrierung des Eingangspunkts 66
Abbildung 36: Die Bestandteile der Benutzeroberfläche von Eclipse 67
Abbildung 37: Das Komponentendiagramm einiger Subystemen 68
Abbildung 38: Das Klassendiagramm der Komponente 73
Abbildung 39: Die Komponente und die mit ihr verbundenen Elemente 74
Abbildung 40: Der Konnektor und die Name-Box und Dummy Figuren von ihm 75
Abbildung 41: Das Klassendiagramm des Konnektors 76
Abbildung 42: Das Klassendiagramm von Palette 77
Abbildung 43: Das Klassendiagramm des Editors 78
Abbildung 44: Der Aufbau des Teilsystems Dateioperationen 80
Abbildung 45: Das erzeugte Diagramm 82
Abbildung 46: Das DiagrammBean Klassendiagramm 83
Abbildung 47: Das Klassendiagramm von den Readers und Writers 85
Abbildung 48: Ein Beispiel für OT 90
Abbildung 49: Mögliche Ursachen der Inkonsistenz 92
Abbildung 50: Veröffentlichen und das Aktualisieren 92
Abbildung 51: Unterschiede zwischen Master und Slave 93
Abbildung 52: Kommunikationen zwischen den Benutzer bei der Kollaboration 94
Abbildung 53: „Gleichzeitige“ Veröffentlichung 94
Abbildung 54: Aufbau des Subsystems Kollaboration 95

Listings

Listing 1: Die Generierung einer Diagramm-URL für den Editor 79

Listing 2: Die Diagram-Datei eines Diagramms 81

Listing 3: Die erzeugte HTML-Seite eines Diagramms 82

Listing 4: Durch den HistoryShower generiertes JavaScript 98

Abkürzungsverzeichnis

ACL	Access Control List
ACM	Access Control Matrix
ADL	Architekturbeschreibungssprache
AJAX	Asynchronous JavaScript and XML
AOP	Aspektorientierte Programmierung
API	Application Programming Interfaces
ASP	Active Server Page
ATAM	Architecture Trade-off Analysis Method
AWT	Abstract Window Toolkit
BPMN	Business Process Modeling Notation
C&C	Component-And-Connector
CRM	Customer Relationship Management
CVS	Concurrent Versions System
DAC	Discretionary Access Control
DOM	Document Object Model
DOM4J	Document Object Model For Java
DSL	Domänenspezifische Sprache
EJB	Enterprise JavaBean
GEF	Graphical Editing Framework
GMF	Graphical Modeling Framework
GROVE	GRoup Outline Viewing Edit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrierte Entwicklungsumgebung
IDL	Interface Description Language
JDOM	Java-based Document Object Model

JFC	Java Foundation Classes
JNI	Java Native Interface
MAC	Mandatory Access Control
MDA	Modellgetriebene Architektur
MVC	Model View Controller
OMG	Object Management Group
OSGi	Open Service Gateway Initiative
OT	Operational Transformation
PDE	Plugin Development Enviroment
PHP	Personal Home Page
QOC	Questions, Options, and Criteria
RAP	Remote Application Platform
RBAC	Role Based Access Control
RM-ODP	The Reference Model of Open Distributed Processing
RSA	Rational Software Architect
SAX	Simple API For XML
SEI	Software Engineering Institute of Carnegie Mellon University
SOA	Service-Oriented Architecture
SVG	Scalable Vector Graphics
SWA	Forschungsgruppe Software Architecture
SWT	Standard Widget Toolkit
TOGAF	The Open Group Architecture Framework
VbMF	View-based Modeling Framework
VML	Vector Markup Language
VML	Vending Machine Languag
VSP	Virtual Service Plattform
W3C	The World Wide Web Consortium
WebVbMF	Web- and View-based Modeling Framework
WICSA1	The First Working IEEE/IFIP Conference on Software Architecture

WOTE	Write Once, Test Everywhere
WSDL	Web Services Description Language
WYSIWIS	What You See Is What I See
UML	Unified Modeling Language
URL	Universal Resource Locator
XSD	XML Schema Definition
Xpath	XML Path Language
XSLT	Extensible Stylesheet Language Transformations

1 Einleitung

1.1 Motivation

Das Betriebssystem Windows XP wurde im Jahre 2001 von Microsoft veröffentlicht. Die Installationsdateien von Windows XP können auf einer CD gespeichert werden, somit ist die Größe der Installationsdateien kleiner als 650 MB, die Kapazität einer CD. Im Jahr 2009 hat Microsoft das Betriebssystem Windows 7 veröffentlicht, die Installationsdateien dieses Betriebssystems sind so groß (über 3GB), dass sie nicht mehr auf einer CD gespeichert werden können. Mit dieser Größenveränderung ist Windows 7 stabiler, schöner, einfacher und leistungsfähiger als Windows XP. Das Gleiche gilt auch für die Entwicklung anderer Betriebssysteme wie Mac OS, Ubuntu usw. sowie für die Entwicklung von vielen anderer Anwendungssoftware. Deshalb ist es nicht schwer zu beobachten, dass Software immer größer und komplexer wird. Der Grund dafür sind die immer vielfältigeren und strengeren Anforderungen an Software, wie Performance, UI-Design usw. Dies macht die Softwarearchitektur immer wichtiger für den Entwurf und die Implementierung heutiger Software. Zusätzlich dazu ist die Softwarearchitektur dringend notwendig wenn die Komplexität der zu entwickelnden Software eine bestimmte Grenze überschreitet.

Die Erstellung der Softwarearchitektur ist eine zeit- und kostenintensive Arbeit. Wenn die Softwarearchitektur nicht wiederverwendet werden kann, wird der Wert der Softwarearchitektur offensichtlich dadurch beeinträchtigt. Die Voraussetzung für die Wiederverwendung von Softwarearchitektur ist dass sie dokumentiert werden kann. Deshalb ist die Frage, wie die Softwarearchitektur dokumentiert werden soll, für die Softwareindustrie sehr wichtig ist.

Es ist auch nicht schwer zu entdecken, dass der Fokus auf das Web und Kollaboration steigende Bedeutung in den Softwareindustrien gewinnt.,Mmehr und mehr Unternehmen in der Softwarebranche unterstützen jetzt webbasierte Kollaboration, z.B. die Plattform

OneDrive[1] von Microsoft ermöglicht den Benutzern Office-Dateien gemeinsam im Internet zu bearbeiten. Die Creative Cloud[2] von Adobe bietet die Möglichkeit, fast alle Produkte von Adobe kollaborativ zu nutzen.

1.2 Zielsetzung

Aufgrund der in der Motivation erläuterten Tatsachen wird in dieser Arbeit die Dokumentation von Softwarearchitektur untersucht. Da die Darstellung in Diagrammform in der Softwarebranche weit verbreitet ist, konzentriert sich diese Arbeit nur darauf, wie die Softwarearchitektur mittels Diagrammen zu dokumentieren.

Zusätzlich dazu beschäftigt sich diese Arbeit auch damit, die Implementierung des Projekt WebVbMF vorzustellen. Dieses Projekt dient als praktischer Teil dieser Arbeit und soll einen webbasierten und kollaborativen Editor, der einige Bestandteile der Softwarearchitektur als Diagramme dokumentieren kann, realisieren. Durch die Implementierung dieses Projektes sollen das Ergebnis der theoretischen Forschung dieser Arbeit überprüft werden sowie die Möglichkeiten, Softwarearchitektur kollaborativ zu dokumentieren, untersucht werden.

Zusammenfassend lautet die Forschungsfrage dieser Arbeit: Wie kann die Softwarearchitektur mittels Diagramme dokumentiert werden und ist es möglich, bei der Dokumentation im Internet/Intranet kollaborativ zu arbeiten?

1.3 Aufbau der Arbeit

Der Hauptteil dieser Arbeit gliedert sich in vier Kapitel. (Kapitel 2 bis Kapitel 5) Das zweite Kapitel beschäftigt sich mit der Vorstellung des Begriffs Softwarearchitektur. In diesem Kapitel sollen die Geschichte der Softwarearchitektur vorgestellt werden, die Frage, warum ist Softwarearchitektur für die Entwicklung einer Software wichtig, wird auch in diesem Kapitel beantwortet.

[1] https://onedrive.live.com/about/de-de/
[2] https://www.adobe.com/at/creativecloud.html

Das dritte Kapitel untersucht die Dokumentation von Softwarearchitektur. Da die Softwarearchitektur-Dokumentationsartefakte unterschiedliche Formen aufweisen können, werden zuerst diese Formen vorgestellt. Die Mit der Dokumentation von Softwarearchitektur im Zusammenhang stehenden Kenntnisse, wie die Softwarearchitektursichten, die möglichen Architekturstile, die Architekturmodelle werden anschließend vorgestellt. Am Ende dieses Kapitels wird erläutert, wie eine Softwarearchitektur mittels Diagrammen dokumentiert werden kann.

Das vierte Kapitel beschäftigt sich mit der Vorstellung der graphischen Architektur-Tools, die Eigenschaften solcher Tools und deren Unterschiede sind der Schwerpunkt dieses Kapitels.

Im fünften Kapitel wird die Implementierung von WebVbMF vorgestellt. Die Schwerpunkte des Projekts sind der Editor und die Unterstützung von Kollaboration, sie werden daher ausführlich vorgestellt. Alle anderen wichtigen Probleme bei der Implementierung wie die Webbasiertheit, die Dateioperationen, die Persistenz, die Zugriffskontrolle werden ebenfalls vorgestellt.

2 Softwarearchitektur

2.1 Geschichte

Nur etwa 20 Jahre nach der Entstehung von Computern ist die Beschäftigung mit Softwarearchitektur mit dem Auftauchen der ersten größeren Softwaresysteme aufgetreten. Solche Systeme machten es notwendig, die gesamten Entwicklungsaufgaben aufzuteilen und die Beziehungen zwischen den Entwicklungseinheiten zu definieren. Der Begriff Softwarearchitektur findet sich aber erst im Jahr 1969 im Tagungsband einer Konferenz über Softwaretechnik [35,54].

Im Jahr 1972 hat David Parnas einen einflussreichen Artikel veröffentlicht. Obwohl er dabei den Begriff Softwarearchitektur nicht verwendet hat, hat er einige Konzepte und Ideen zur Softwarearchitektur dargestellt. Die Schlüsselkonzepte zum Entwurf und der Organisation von Softwaresystemen, welche die Bestandteile von Softwarearchitektur sind, wurden 1975 im Buch „The Mythical Man Month“ von Frederick P. Brooks diskutiert. In den 1980er Jahren wurden diese Konzepte und Ideen von Parnas und Brooks weiter vertieft [25].

Erst in den 1990er Jahren wurde die Softwarearchitektur ein unabhängiges Teilgebiet der Softwareentwicklung. Im Jahr 1992 haben Dewayne Perry und Alexander Wolf den Artikel „Foundations for the Study of Software Architecture“, in dem sie Softwarearchitektur als die Kombination von Elementen, Form und Begründungen skizziert wird, veröffentlicht. Daraufhin haben Universitäten eine Reihe von Computersprachen für die Beschreibung von Softwarearchitektur entwickelt, z.B. C2, Rapide, Darwin usw. [25].

Seit 1995 hat die Softwarearchitektur sowohl die Aufmerksamkeit der Industrie als auch von akademischen Einrichtungen bekommen, und zahlreiche Theorien über Softwarearchitektur z.B. Architektursichten, Entwurfsmuster usw., wurden entwickelt [54]. Gleichzeitig haben viele Softwarefirmen wie Siemens, Nokia, Philips, IBM usw.

Softwarearchitektur benutzt, um bessere Wiederverwendbarkeit von Software zu erzielen sowie um die Softwareproduktlinien zu entwerfen [25]. Im Jahr 1999 hat die erste internationale Konferenz, The First Working IEEE/IFIP Conference on Software Architecture (WICSA1), zum Thema Softwarearchitektur stattgefunden [46].

ANSI/IEEE 1471-2000[3], welche später im Jahr 2007 als ISO/IEC 42010 übernommen wurde, wurde im Jahr 2000 zur Architekturbeschreibung von Softwaresystemen veröffentlicht. Die Architekturbewertungsmethode Architecture Trade-off Analysis Method (ATAM) wurde später vom Software Engineering Institute (SEI) der Carnegie Mellon University entwickelt. Mit ATAM kann man herausfinden, ob Softwarearchitektur bestimmte Qualitätsziele erfüllt hat sowie wie die Qualitätsziele miteinander verbunden sind [2]. Seit 2003 unterstützt die Unified Modeling Language (UML) auch die Dokumentation von Softwarearchitektur [13].

Die obigen Bestrebungen bilden die Geschichte der Softwarearchitektur ab. Man kann die Geschichte der Softwarearchitektur in drei Phasen unterteilen. Die erste Phase beginnt im Jahr 1960 und endete im Jahr 1990, dies ist die Anfangsphase der Softwarearchitektur. Die zweite Phase, die Etablierungsphase, entspricht dem Zeitraum von 1990 bis 2000. Die aktuelle Entwicklungsphase der Softwarearchitektur beginnt im Jahr 2000 [25,54]. Heute konzentriert sich die Softwarearchitektur auf serviceorientierte Architekturen, Cloud Computing, mobile Anwendungen, Wissensmanagement für Softwarearchitekturen, usw. Solche Praxis- und Forschungsthemen entsprechen genau den populären Trends in der Softwareindustrie [43].

Nach dieser kurzen Geschichte wird der Begriff Softwarearchitektur im folgenden Abschnitt erläutert.

[3] Recommended Practice for Architecture Description of Software-Intensive Systems

2.2 Begriff

Es ist nicht schwer zu beobachten, dass die Software immer größer und komplexer wird. Dies ist die unvermeidliche Konsequenz der Entwicklung von Software, da die Anforderungen an Software (z.B. Performance, Sicherheit, Wiederverwendbarkeit usw.) immer vielfältiger und strenger werden. Ohne Softwarearchitektur ist die Entwicklung solcher Software unpraktikabel [3,7,13,36,54].

Obwohl Softwarearchitektur eine wichtige Rolle im Softwareentwicklungsprozess spielt gibt es keine Einigkeit über die Bedeutung des Begriffs Softwarearchitektur. Heute gibt es viele unterschiedliche Definitionen für Softwarearchitektur, beispielsweise hat das Software Engineering Insititute der Carnegie Mellon University schon über 200 verschiedene Definitionen von Softwarearchitektur gesammelt [38]. Trotzdem sind diese Definitionen sehr ähnlich und die Unterschiede dazwischen grundsätzlich durch unterschiedliche Blickwinkel verursacht. Meiner Meinung nach kann folgende Definition den Begriff Softwarearchitektur gut erklären:

> *„The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both [3].“*

Laut Bass, L. et al. müssen folgende Punkte beachtet, um diese Definition richtig zu verstehen:

- **Die Beziehung zwischen der Softwarearchitektur und der Struktur von Software.** Die Struktur von Software besteht aus den Softwareelementen und deren Beziehungen. Die Software besteht wiederum aus Strukturen. Die Strukturen, die die Eigenschaften der Software beschreiben und für das logische Denken der Software wichtig sind, stellen die Softwarearchitektur dar. Das heißt, dass manche Strukturen kein Bestandteil der Softwarearchitektur sind. Normalerweise können nur folgende Arten von Strukturen zur Softwarearchitektur zählen:

- ✓ Die Strukturen, die das gesamte System in Teile, die als Implementierungseinheiten geeignet sind, zerlegen. Solches Teil, das Modul genannt ist, kann zu einer Person oder einem Team zugewiesen. Beispielsweise unterteilt eine Struktur ein System in Datenbank, Benutzeroberfläche, Geschäftsprozess usw. und jedes Teil davon wird von einem entsprechenden Team übernommen, diese Struktur ist Bestandteil von Softwarearchitektur.

- ✓ Die Strukturen, die die Interaktionen zwischen den Softwareelementen in der Laufzeit beschreiben. Diese Art von Strukturen skizziert die dynamischen Eigenschaften des Systems. Solche architektonischen Strukturen nennen wir Component-And-Connector (C&C) Strukturen.

- ✓ Die Strukturen, die beschreiben, wie die anderen Strukturen zur Plänen bezüglich Organisierung, Implementierung, Installation, Ausführung usw. des Systems transformiert sollen. Diese architektonischen Strukturen sind Allokationsstrukturen. Beispielsweise kann eine solche Art von Strukturen bestimmen, welche Implementierungseinheit von welcher Person oder welchem Team übernommen werden soll, welche Komponente auf welches System installiert werden soll, usw.

Insgesamt kann man über die Beziehung zwischen der Softwarearchitektur und der Softwarestruktur sagen, dass die Softwarearchitektur eine Sammlung von wichtigen, nicht allen, Strukturen ist, und die einzelne Struktur allein nicht als Softwarearchitektur betrachtet kann [3].

- **Die Softwarearchitektur ist eine Art von Abstraktion.** Da die Softwarearchitektur besteht aus den Strukturen, die wiederum aus den Softwareelementen und den Beziehungen daziwschen bestehen, kann die Softwarearchitektur als eine Sammlung von den Softwareelementen und deren Beziehungen betrachtet. Aber die Softwarearchitektur konzentriert sich nur auf

die wichtigen Softwareelemente und Beziehungen, alle anderen Softwareelemente und deren Beziehungen werden weggelassen. In diesem Sinne ist die Softwarearchitektur eine Abstraktion des zu entwickelnden Systems. Diese Abstraktion kann zur Handhabung der Komplexität moderner Systems beitragen [3].

- **Jedes Softwaresystem hat eine Softwarearchitektur.** Weil das Softwaresystem aus den Softwareelementen und deren Beziehungen besteht, gibt es immer Softwareelemente und Beziehungen, die für das Softwaresystem wichtig sind und daher eine Softwarearchitektur abbilden können. Wenn das System trivial ist, kann das gesamte System als ein Softwareelement betrachtet werden. Mit diesem einzigen Softwareelement kann auch eine Softwarearchitektur abgeleitet werden, obwohl diese Softwarearchitektur nicht sinnvoll genutzt werden kann [3,36].

- **Die Softwarearchitektur inkludiert das Verhalten der Softwareelemente.** Das Verhalten der Softwareelemente ist die Interaktionen zwischen den Softwareelementen. Solche Interaktionen werden als Beziehungen zwischen den Softwareelementen in der Definition von Softwarearchitektur integriert. Die Interaktionen, die die Funktionalität des Systems bereitstellen, sind vielfältig, sie können z.B. Daten austauschen, Anmeldungen weiterleiten, die Verfügbarkeit anderen Elemente beeinflussen usw. Aber die Softwarearchitektur kümmert sich nicht um alle Beziehungen, nur um solche, die Auswirkungen auf die anderen Elemente oder das gesamte System hat. Wenn das Verhalten eine kleine Granularität aufweist, ist es nicht tgeignet in die Softwarearchitektur aufgenommen zu werden, da die Ebene, die dieser Granularität entspricht, zu niedrig für die Softwarearchitektur ist [3].

Allgemein kann Softwarearchitektur als eine Sammlung von für das logische Verständnis des Softwaresystems wichtigen Softwarestrukturen, die aus Softwareelementen und den Beziehungen dazwischen bestehen, betrachtet werden [3,54,55]. Nächste zu erklärende Frage ist, warum die Softwarearchitektur wichtig ist.

2.3 Bedeutung

Die Softwarearchitektur ist für das Softwaresystem wichtig. Laut Bass, L. et al. sind die Gründe dafür:

- **Die Softwarearchitektur kann bestimmen, welche Qualitätsattribute eines Systems gehemmt oder aktiviert werden sollen.** Die Qualitätsattribute eines Systems sind z.B. Verfügbarkeit, Interoperabilität, Modifizierbarkeit, Leistung, Sicherheit, Testbarkeit, Verwendbarkeit usw. Wegen der Diversität der Qualitätsattribute ist es schwer für die meisten Softwaresysteme, die Anforderungen an all diese Qualitätsattribute gleichzeitig zu erfüllen. Deshalb muss bei der Entwicklung eines Softwaresystems zuerst auf oberster Ebene überlegen, welche Qualitätsattribute für das System wichtig und welche trivial sind. Diese Aufgabe wird grundsätzlich im Rahmen der Softwarearchitektur erfüllt. Aber die Softwarearchitektur selbst kann nicht garantieren, dass die erwarteten Qualitätsattribute erreicht werden können, da die Qualitätsattribute durch andere Faktoren z.B. die Codierung und die Deployment beeinflusst werden können. Deshalb sind entsprechende Maßnahmen notwendig, um die im Rahmen der Softwarearchitektur festgestellten Qualitätsattribute zu erreichen [3,54]. Wenn z.B. Sicherheit bei der Softwarearchitektur als das wichtigste Qualitätsattribut bestätigt ist, muss die Kommunikationen zwischen den Softwareelementen verschlüsselt und kontrolliert werden. Einige zusätzliche Mechanismen wie Zugriffskontrolle und Authentifizierung sind normalerweise auch wesentlich.

- **Die Softwarearchitektur kann die Veränderungen des Softwaresystems beurteilen und verwalten.** Eine Statistik zeigt, dass etwa 80 Prozent der gesamten Kosten eines Softwaresystems durch die Aktivitäten zur Behandlung von Veränderungen nach der Erstbereitstellung verursacht werdem. Deshalb ist das Qualitätsattribut Modifizierbarkeit für die Kosten des Softwaresystems wichtig. Die Veränderungen des Softwaresystems lassen sich drei Arten

zuordnen:

- ✓ **Lokale Veränderungen.** Diese Veränderung kann durch Modifizierung einzigen Softwareelementes durchgeführt werden. Beispielsweise das Aktualisieren der in der Datenbank gespeicherten Preise.

- ✓ **Nichtlokale Veränderungen.** Um solche Veränderungen durchzuführen muss eine Gruppe von Softwareelementen aber nicht die Softwarestruktur modifizieren werden. Beispielsweise wenn eine neue Preislogik eingeführt wird, müssen dementsprechend die Datenbank und das Modul, das die Preise berechnet, modifiziert werden.

- ✓ **Architektonische Veränderungen.** Diese Art von Veränderungen beeinflussen die Logik des Systems oder die grundlegende Komunikation zwischen den Softwareelementen. Als Konsequenz muss vielleicht das ganze System modifiziert werden. Beispielsweise die Transformation einer lokalen Anwendung zu einer web-basierten Anwendung.

Die Softwarearchitektur kann dazu beitragen, die Art und die Wichtigkeit der Veränderungen zu bestimmen, die Veränderung, die niedrigsten Kosten mit sich verbunden ist, herauzufinden, die Konsequenzen einer Veränderung vorherzusagen usw. Wenn mehre Veränderungen in das System eingesetzt müssen, die Softwarearchitektur kann auch die Reihenfolge und die Prioritäten der Veränderungen geben [3].

- ➢ **Die Softwarearchitektur kann die Qualität des zu entwickelnden Systems prognostizieren.** Die architektonischen Entscheidungen können entsprechende Qualitätsattribute verursachen. Deshalb kann die Qualität eines Systems vor der Implementierung durch die Analyse seiner Softwarearchitektur, konkret indem man die architektonischen Entscheidungen der Softwarearchitektur prüft, prognostiziert werden [3,54].

- **Die Softwarearchitektur kann die Kommunikationen zwischen den Stakeholdern verbessern.** Wie schon erwähnt wurde, die Softwarearchitektur ist eine Art von Abstraktion und eine Sammlung von Softwarestrukturen. Diese beiden Eigenschaften ermöglichen das bessere Verständnis für das System. Die nichttechnischen Strukturen sind für Stakeholder wie Benutzer, Kunden und Projekt Manager geeignet, dagegen sind die technischen Strukturen für die Stakeholder wie Programmierer und Tester notwendig. Das heißt, mit Hilfe der Softwarearchitektur können unterschiedliche Stakeholder das System verstehen. Dann können sie über das System diskutieren und das System kann dadurch verbessert werden [3,54].

- **Die Softwarearchitektur bringt die frühen Entwurfsentscheidungen.** Die Softwarearchitektur, also die Sammlung der Softwarestrukturen, ist der Entwurf eines Systems. Der Entwurf kann als eine Reihe von Entscheidungen betrachtet werden. Deshalb kann die Softwarearchitektur als eine Sammlung von Softwarestrukturen betrachtet werden. Solche Entscheidungen sind für die verbleibenden Aufgaben wie Implementierung und Deployment usw. wesentlich. Wenn diese Entscheidungen bei der Implementierung oder Deployment verändert werden, können sehr wohl hohe Kosten entstehen oder das gesamte System beeinflusst werden [3]. Zu den frühen Entwurfsentscheidungen gehören z.B. Soll das System zentral oder verteilt sein? Welches Betriebssystem wird benutzt? Müssen die Informationen verschlüsselt werden oder nicht?

- **Die Softwarearchitektur begrenzt die Implementierung.** Die Implementierung soll das in der Softwarearchitektur definierte System realisieren. Die Voraussetzung für die richtige Realisierung des Systems ist es, die in der Softwarearchitektur getroffenen Entwurfsentscheidungen immer zu verfolgen. Das heißt, bei der Implementierung müssen immer die vordefinierten Softwareelemente implementiert werden, es müssen die Softwareelemente immer mittels der vordefinierten Kommunikationswege miteinander interagieren sowie müssen die Softwareelemente immer die vordefinierten Funktionen haben.

Deshalb liefert die Softwarearchitektur Beschränkungen für die Implementierung [3].

- **Die Softwarearchitektur beeinflusst die Organisationsstruktur.** Ein wichtiger Bestandteil der Softwarearchitektur ist die Struktur, die das gesamte System in Teile, die als Implementierungseinheiten geeignet sind, zerlegt. Dementsprechend ist es innerhalb eines Projektes üblich, das gesamte Personal in unterschiedliche Teams aufzuteilen, sodass jedes Team sich auf bestimmte Teilaufgaben des Projektes konzentrieren kann. In diesem Sinne dient die Softwarearchitektur als die Grundlage für die Personalallokation [3].

- **Die Softwarearchitektur ermöglicht das evolutionäre Prototyping.** Wenn die Softwarearchitektur definiert ist, kann ein Skelettsystem vor der Implementierung des gesamten Systems aufgebaut werden. In einem Skelettsystem werden zumindest einige Elemente des zu entwickelnden Systems implementiert. Beispielsweise kann ein Skelettsystem die Initialisierung von Elementen, den Zugriff auf Dateien usw. realisieren. Das Skelettsystem kann das potenzielle Risiko sinken. Wenn das zu entwickelnde System zu einer Produktfamilie gehört, können mithilfe von Skelettsystem die gesamten Kosten der Produktfamilie sinken [3].

- **Die Softwarearchitektur ermöglicht die besseren Schätzung der Kosten und des Zeitplasn.** Die geeignete Schätzungen der Kosten und des Zeitplans sind für den Projektmanager wichtig, da er die entsprechenden Ressourcen zur Verfügung stellen muss sowie den ganzen Entwicklungsprozess kontrollieren muss. Die aufgrund des Verständnisses der Softwareelemente des Systems geschätzten monetären und zeitlichen Kosten sind präziser als die Kosten, die nur aufgrund des gesamten Systems geschätzt sind. Deshalb dient die Softwarearchitektur als die Basis für die besseren Schätzungen der Kosten und des Zeitplans. Aber das bedeutet nicht, dass man für die Schätzungen nur die Softwarearchitektur braucht. Um die präzisesten Schätzungen zu bekommen muss nicht nur die

Softwarearchitektur sondern auch auf oberster Ebene das gesamte System berücksichtigt werden [3,54].

- **Die Softwarearchitektur liefert ein wiederverwendbares Modell.** Je früher die wiederverwendbare Gegenstände in das System eingeführt, desto mehr Vorteile lassen sich erzielen. Deshalb muss man zuerst überlegen, ob es eine wiederverwendbare Softwarearchitektur, die für das zu entwickelnde System geeignet ist gibt, da die Wiederverwendung von Softwarearchitektur die meisten Vorteile bringen kann. Dann die Frage ist, ob die Softwarearchitektur wiederverwendbar ist. Die Antwort ist „Ja“. Die Anforderungen an das System, welche zu einer Softwarearchitektur führen, die während der Softwarearchitektur gesammelten Erfahrungen und das aufgrund der Softwarearchitektur aufgebaute Skelettsystem sind grundsätzlich wiederverwendbar. Außerdem hat die Wiederverwendung von Softwarearchitektur für die Produktlinie und die Produktfamilie besondere Bedeutung. Eine Produktlinie oder Produktfamilie ist eine Menge von Systemes, welche bei der Implementierung die gleiche Menge an Assets benutzen. Eine sogenannte Produktliniearchitektur bestimmt, was für alle Mitglieder fix ist sowie was variabel ist. Wenn die Softwarearchitektur eines konkreten Produkts vorhanden ist, sind die von der Produktliniearchitektur definierten fixen Teile dieser Softwarearchitektur für alle anderen Mitglieder wiederverwendbar [3,54].

- **Die Softwarearchitektur stellt die Einführung von fremden Komponenten sicher.** Die Einführung von fremden Komponenten ist denkbar, da dadurch das System schneller auf den Markt gebracht werden kann, die Zuverlässigkeit des Systems erhöht werden kann, die Entwicklungskosten gesunken werden können sowie die Flexibilität des Implementierungsprozesses erhöht werden kann. Wie schon erläutert wurde, kann die Softwarearchitektur die Implementierung begrenzen, das heißt, dass die Softwarearchitektur bestimmte Einschränkungen, z.B. die Interaktionswege zwischen den Softwareelementen und der Umgebung, die Inputs und die Outputs der Softwareelemente für die Implementierung

definiert. Wenn eine Komponente die von der Softwarearchitektur definierten Einschränkungen erfüllen kann, kann sie in das System eingeführt werden. Die Herkunft dieser Komponente ist egal, es kann sich z.B. um Open Source, öffentlich verfügbare Anwendungen oder selbstentwickelte unabhängige Komponente handeln [3].

- **Die Softwarearchitektur beschränkt die Möglichkeiten des Entwurfs.** Wenn man ein System entwirft, gibt es viele Möglichkeiten. Beispielsweise können die Softwareelemente, aus denen das System bestehen kann, kombiniert werden. Dadurch werden unterschiedliche Kombinationen erzeugt. Viele Kombinationen können die Anforderungen an das System erfüllen. Um die Komplexität zu kontrollieren muss die Menge der Möglichkeiten auf ein bestimmtes Niveau gesenkt werden. Dies entspricht genau der Disziplin des Software Engineerings, nämlich, um besseres Ergebnis zu schaffen müssen die Möglichkeiten beschränkt werden. Die Softwarearchitektur, wie schon erläutert hat, definiert bestimmte Einschränkungen und kann daher die Möglichkeiten des Entwurfs beschränken [3,54].

- **Die Softwarearchitektur ist die Basis für das Training.** Die Softwarearchitektur, die die Softwareelemente und deren Interaktion darstellt, kann als die erste Einführung in das System für das Training von neuen Projektmitgliedern verwendet werden. Außerdem ist die Softwarearchitektur für das Verständnis der Systemstruktur, der Personalallokation und das Verhalten des Systems nützlich [3].

Die grundlegenden Kenntnisse der Softwarearchitektur, also die Geschichte, den Begriff und die Bedeutung, sind in diesem Kapitel vorgestellt. Da der Schwerpunkt dieser Masterarbeit liegt darauf, die Softwarearchitektur zu dokumentieren, wird die Dokumentation der Softwarearchitektur im folgenden Kapitel erklärt.

3 Dokumentation der Softwarearchitektur

Die Vorteile von der Softwarearchitektur wurden im letzten Kapitel vorgestellt. Eine wichtige und allgemeine Voraussetzung für die Erzielung dieser Vorteile ist die umfassende Dokumentation der Softwarearchitektur. Ohne diese Dokumentation wird die Softwarearchitektur grundsätzlich nutzlos, da die Leute, die sie brauchen, nicht genug über sie wissen. Als Konsequenz können die Leute sie nicht vollständig verstehen und daher das gewünschte System nicht implementieren, im schlimmsten Fall können die Leute sie missverstehen und das gewünschte System falsch implementieren. Ohne diese Dokumentation der Softwarearchitektur wird außerdem die Kommunikationen zwischen den Stakeholdern verhindert [7].

Im diesem Kapitel werden zuerst die möglichen Dokumentationsformen vorgestellt. Weil die Softwarearchitektursichten und der Architekturstil für die Dokumentation sehr wichtig sind, werden sie dann und die mit den Softwarearchitektursichten im Zusammenhang stehenden Architekturmodelle erklärt. Am Ende dieses Kapitels steht die Vorstellung des Weges zur graphischen Dokumentation von Softwarearchitektur.

3.1 Formen

Die Dokumentation der Softwarearchitektur kann unterschiedliche Formen aufweisen. Laut Clements, P. et al. sind diese Formen:

- **Natürliche Sprachen.** Eine natürliche Sprache ist eine von Menschen gesprochene oder gebärdete Einzelsprache [20]. Beispielsweise kann eine Softwarearchitektur wie folgt textuell beschrieben werden: Dieses System besteht aus zwei Komponenten, Bestellung und Zahlen, die durch den Konnektor Zahlungsmethode miteinander kommunizieren. Wenn eine Softwarearchitektur mittels einer natürlichen Sprache dokumentiert ist, ist sie nur für die Leute, die diese Sprache sprechen, verständlich. Darüber hinaus ist diese Form nicht übersichtlich, sodass die Struktur und die Logik der Softwarearchitektur nur

schwer zu entdecken ist. Deshalb ist diese Form nur für sehr einfache Softwarearchitektur und zur Ergänzung der Softwarearchitektur anwendbar [3,7].

- **Bilder.** Die Elemente und die Beziehungen, die ein Bild verwendet, sind nicht universal [3,7]. Folgende Abbildung zeigt ein Beispiel für solch eine Form der Dokumentation.

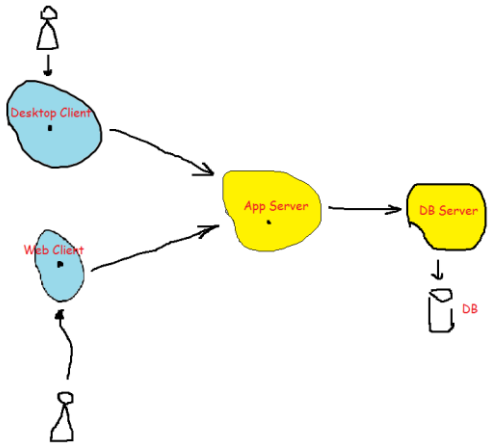

Abbildung 1: Dokumentation der Softwarearchitektur als Bild

Es ist schwierig, die Softwarearchitektur in Form von Bildern außerhalb der Entwicklungsorganisation zu bereitstellen. Trotz besserer Verständlichkeit und besserer Übersichtlichkeit als natürliche Sprache sind Bilder wie die natürlichen Sprache nur für sehr einfache Softwarearchitekturen anwendbar [3].

- **Formalen Sprachen.** Eine formale Sprache ist eine abstrakte Sprache, bei der nicht die Kommunikation sondern die mathematische Verwendung im Vordergrund steht [7,29]. Beispielsweise sind domänenspezifische Sprachen (DSLs) formale Sprachen. Die Vending Machine Language (VML), die von Matthias Anlauff und Asuman Sünbül entwickelt wurde, ist eine DSL für die Dokumentation der Softwarearchitektur [1].

- **Architekturbeschreibungssprachen (ADLs).** Die ADLs sind speziell für die Dokumentation der Softwarearchitektur entwickelt worden. Sie verwenden Grafiken und Texte um den Inhalt einer Softwarearchitektur zu dokumentieren

[3,7,23,54]. Viele unterschiedlichen Anwendungen, wie Acme[4] der Carnegie Mellon University, byADL[5] der Universit à degli Studi dell'Aquila, LePUS3 and Class-Z[6] der University Of Essex usw. sind für die Erstellung, Modifizierung, Simulation und Analyse der ADLs anwendbar.

- **Diagramme.** Ein Diagramm ist eine grafische Darstellung von Informationen, Daten oder Sachverhalten. Viele Arten von Diagrammen wie Klassendiagramm, ER-Diagramm, Prozessdiagramm usw. sind universal. Das heißt, die Elemente, die Beziehungstypen solcher Arten von Diagrammen sind vordefiniert und daher weltweit anwendbar. Die Softwarearchitektur kann mittels Diagrammen präzis dokumentiert. Die Diagramme sind in der Tat die wichtigste und am häufigsten verwendete Form für die Dokumentierung von Softwarearchitektur und werden in den folgenden Kapiteln vertiefend diskutiert [7,54].

Diese Formen schließen sich nicht gegenseitig aus. Das heißt, unterschiedliche Formen können innerhalb einer Softwarearchitektur gleichzeitig benutzt werden.

3.2 Sichten

Betrachten wir zuerst ein Beispiel. Der menschliche Körper, der aus vielen unterschiedlichen Subsystemen wie Skelettsystem, Nervensystem, Gefäßsystem usw. besteht, ist ein komplexes System. Der Orthopäde, der Neurologe und der Hämatologe konzentrieren sich auf verschiedenes Subsysteme und daher haben unterschiedliche Sichten auf den menschlichen Körper. Obwohl diese Sichten sehr unterschiedlich aussehen (vgl. Abbildung 2), stehen sie miteinander in Zusammenhang. Jede einzelne Sicht ist keine Architektur des menschlichen Körpers, da sie nur ein Subsystem, nicht das Gesamtsystem beschreiben kann. Aber sie trägt bestimmte Informationen über das Gesamtsystem und daher ist ein Bestandteil des Gesamtsystems [3].

[4] http://www.cs.cmu.edu/~acme/
[5] http://byadl.di.univaq.it/
[6] http://www.lepus.org.uk/

Abbildung 2: Einige Subsysteme des menschlichen Körpers

(Quell: www.zygotebody.com)

Das Gleiche gilt für die Softwarearchitektur. Die Softwarearchitektur ist eine komplexe Entität, die nicht einfach beschrieben werden kann. Deshalb ist es notwendig, die Aufmerksamkeit zu einem beliebigen Zeitpunkt auf bestimmte Seiten der Softwarearchitektur zu beschränken. Dadurch werden unterschiedliche Sichten der Softwarearchitektur abgeleitet. Die Dokumentation dieser Sichten ist einfacher als die Dokumentation der Softwarearchitektur, und durch die Dokumentation dieser Sichten kann die Softwarearchitektur dokumentiert werden.

Eine Sicht der Softwarearchitektur umfasst also eine Seite der Softwarearchitektur. Sie ist daher eine Repräsentation eine kohärentes Menge architektonischer Elementen sowie deren Beziehungen [3,36].

Im Abschnitt 2.2 wurden die Arten der architektonischen Strukturen vorgestellt. Die architektonischen Strukturen lassen sich also in Modulstruktur, Allokationsstruktur und Component-And-Connector Struktur unterscheiden. Da jede dieser Strukturen eine Seite der Softwarearchitektur darstellt, können sie auch als Softwarearchitektursichten betrachtet werden. In Wirklichkeit ist die Klassifikation der Sichten mit der Klassifikation der architektonischen Strukturen identisch [3,54]. (vgl. Abbildung 3)

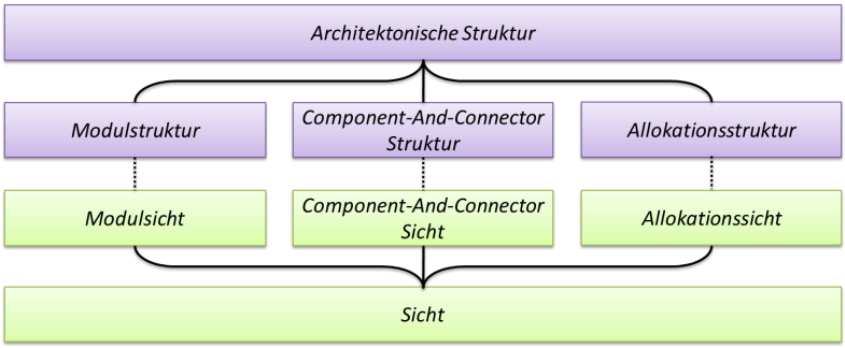

Abbildung 3: Die Klassifikation von architektonischen Strukturen und Sichten

Die Elemente einer Modulsicht sind Module. Als Modul ist eine Implementierungseinheit des Softwaresystems, die ein Menge von kohärenten Aufgaben erfüllen kann, zu verstehen. Ein Modul kann unterschiedliche Eigenschaften wie Name, Verantwortung, Sichtbarkeit der Schnittstelle(n) und Implementierungsinformationen haben. Die Eigenschaft Verantwortung kann die Rolle des Modules im System verdeutlichen. Die Beziehungen zwischen den Modulen können „Is part of“, „Depends on“ und „Is a“ sein [3,7,13,39,54,55].

Die Elemente einer Component-And-Connector Sicht sind Komponenten und die Konnektoren. Die Komponenten sind die während der Laufzeit existierenden Elemente, wie Prozesse, Objekte, Clients, Server und Datenspeicher. Die Schnittstelle der Komponente nennt man Port. Ein Port definiert eine Stelle für die möglichen Interaktionen einer Komponente mit seiner Umgebung. Die Art eines Portes definiert das an diesem Port erlaubte Verhalten. Beispielsweise kann ein Eingangsport nur eingehende Datenflüsse akzeptieren. Eine Komponente kann gleichzeitig mehre Ports des gleichen Typs haben. Die Konnektoren beschreiben die Wege der Interaktionen, wie Kommunikationsverbindungen, Protokolle, Informationsflüsse und die Zugänge zu gemeinsam genutzten Speichern. Der Startpunkt und der Endpunkt eines Weges sind Ports, die durch einen Konnektor verbunden sind. Ein Konnektor besitzt eine unbestimmte Anzahl von Rollen. Die Rolle des Konnektors bestimmt, wie eine Komponente den Konnektor für die Interaktionen benutzen kann. In einem Client-Server-Diagramm kann beispielsweise ein Client-Server-Konnektor die Rollen von „Invokes-services“ und/oder „Provides-services“ haben. Die Elemente einer

Component-And-Connector Sicht, also die Komponenten und die Konnektoren, können die Eigenschaften von Zuverlässigkeit, Leistung, Ressourcenbedarf, Funktionalität, Sicherheit, Parallelität und Ebene haben. Die Beziehungen innerhalb einer Component-And-Connector Sicht sind „Attachment“ und „Interface Delegation“. „Attachment“ gibt an, welche Konnektoren mit welchen Komponenten verbunden sind. Wenn eine Komponente oder ein Konnektor eine interne Struktur hast, wird „Interface Delegation“ benutzt, um die Beziehungen zwischen der internen Struktur und den externen Schnittstellen zu dokumentieren [3,7,39,54].

Die Elemente einer Allokationssicht sind die Softwareelemente und die Elemente der Umgebung. Die Softwareelemente sollen die Eigenschaften, die die Umgebung erfordert, besitzen und vice versa. Die Beziehungen zwischen den Softwareelementen und den Elementen der Umgebung sind „Allocated-to“, dies besagt, dass ein Softwareelement zu einem Element der Umgebung zugewiesen ist [3,7, 39,54].

Die Natur der Dokumentation der Softwarearchitektur ist die Dokumentation relevanter Sichten der Softwarearchitektur. Da jede Sicht nur eine Seite der Softwarearchitektur beschreiben kann und jede Sicht nur das Interesse bestimmter Stakeholdern wiedergeben kann, sind viele unterschiedliche Sichten für die Darstellung der Softwarearchitektur notwendig. Welche Sichten relevant sind, hängt von den vordefinierten Zielen ab [7].

3.3 Architekturmodelle

Weil für die Dokumentation der Softwarearchitektur viele unterschiedliche Sichten notwendig sind, muss man überlegen, welche Sichten analysiert und dokumentiert werden müssen. Wie schon gesagt hängt die Wahl der Sichten von den vordefinierten Zielen ab. Zu den vordefinierten Zielen gehören z.B. die Interessen der Stakeholder, die Struktur des zu realisierenden Softwaresystems usw. Wegen der Komplexität der vordefinierten Ziele ist die Wahl der Sichten eine schwierige Aufgabe. Ein wesentlicher Beitrag der Architekturmodelle zur Dokumentation der Softwarearchitektur ist, dass sie

verdeutlichen, welche Sichten analysiert und dokumentiert werden müssen. Deshalb kann die Aufgabe der Wahl der Sichten mithilfe von Architekturmodellen erleichtert werden [7]. In diesem Abschnitt werden einige Architekturmodelle vorgestellt.

3.3.1 4+1 Sichtenmodell

Im Jahr 1995 hat Philippe Kruchten von Rational Software Corporation einen Artikel „The 4+1 View Model of Software Architecture“ veröffentlicht. In diesen Artikel hat er ein Modell für die Darstellung der Softwarearchitektur präsentiert. Die grundlegende Methode dieses Modells ist es, durch gleichzeitige Verwendung von mehreren Sichten die Interessen unterschiedlicher Stakeholdern zu erfüllen. Dieses Modell hat er 4+1 Sichtenmodell genannt [24].

Das 4+1 Sichtenmodell enthält fünf Sichten (vgl. Abbildung 4), also Logical View, Process View, Development View, Physical View und Scenarios. Alle Sichten dieses Modells können mithilfe von UML graphisch dokumentiert werden [24].

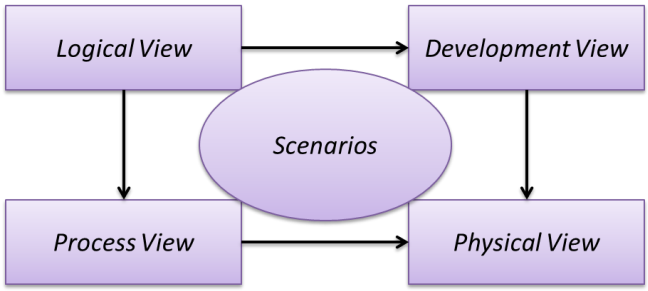

Abbildung 4: Das 4+1 Sichtenmodell

(Quell: www.ibm.com/developerworks/cn/rational/r-4p1-view/)

Die Szenarien sollen die nahtlose Zusammenarbeit der Elemente aller anderen vier Sichten sicherstellen. Die wichtigen Anwendungsszenarien oder Anwendungsfälle sind die Bestandteile dieser Sicht. Diese Sicht ist die wichtigste Sicht des 4+1 Sichtenmodells und kann als redundant zu den anderen Sichte betrachtet werden. Und sie hilft dabei Architekturelemente zu identifizieren, zu veranschaulichen und die Architektur zu überprüfen. Darüber hinaus dient sie als Startpunkt für erste Architekturtests und Implementierungsentwürfe [13,24].

Der Logical View beschäftigt sich mit der Funktionalität des Systems für den Endnutzer. Um solche Sichten zu bekommen wird das System in eine Reihe von Schlüsselabstraktionen zerlegt. Jede Abstraktion enthält eine Menge von zusammenhängenden und mithilfe von Abstraktion, Kapselung und Vererbung abgeleiteten Objekten oder Objektklassen. Diese Sicht dient nicht nur zu funktionellen Analyse sondern auch zu Identifikation des gemeinsamen Mechanismus des Systems und der Designelemente [13,24].

Der Process View beschäftigt sich mit den nichtfunktionalen Anforderungen wie Performance und Verfügbarkeit an das System, also den dynamischen Aspekten des Systems. Diese Sicht soll Parallelität, Verteilung, Integration, Fehlertoleranz sowie das Zusammenspiel von Logical View und Process View darstellen. Nämlich in welchem oder welchen Prozess(en) werden die im Logical View aufgelisteten Objekte durchgeführt [13,24].

Der Development View beschäftigt sich mit dem Softwaremanagement, also mit der Organisation der Module während der Implementierung. Das gesamte Softwaresystem wird in mehre kleine Blöcke (Bibliotheken oder Subsysteme), die von einem oder mehreren Entwicklern entwickelt werden können, verpackt. Die Bibliotheken oder Subsysteme werden schichtweise organisiert und die Bibliotheken oder Subsysteme auf einer Schicht müssen die notwendigen Schnittstellen für die höhere Schicht bereitstellen. Diese Sicht ist die Grundlage für die Aktivitäten während der Implementierung wie z.B. Arbeitszuordnung, Projektüberwachung, Kostenschätzung usw. [13,24].

Der Physical View beschäftigt sich mit der Zuordnung der Softwarekomponenten zur physikalischen Hardware-Teilen. (also der Zuordnung dieser zu Hardware-Teilen) und der Kommunikation zwischen diesen Softwarekomponenten. Diese Sicht ist für verteilte Systeme besonders wichtig [13,24].

Nicht alle Softwarearchitekturen brauchen das 4+1 Sichtenmodell, und nutzlose Sicht kann von den Modell weggelassen werden. Wenn nur ein Prozessor benutzt wird, kann

der Physical View weggelassen werden. Wenn es nur einen Prozess oder ein Programm gibt, kann der Process View weggelassen werden. Für sehr kleines Softwaresystem können der Logical View und der Development View sehr ähnlich sein, dann kann einer davon weggelassen werden. Aber, die Szenarien sind auf jedenfalls anwendbar [24].

Es wird angenommen, dass in einem Unternehmen ein Customer Relationship Management System (CRM) implementiert werden muss. Dieses System soll den Benutzern die Möglichkeiten geben, die Stammdaten sämtlichen Kunden, die Termine sowie die Bestellungen der Kunden zu verwalten [51]. Das Szenario dieses Systems wird in Abbildung 5 gezeigt.

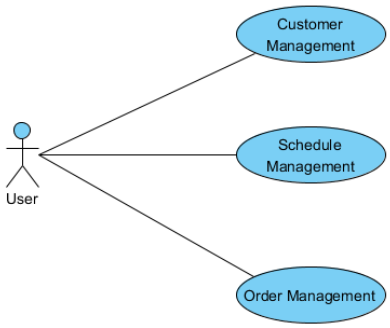

Abbildung 5: Das Szenario eines CRM Systems

Um die Logik des Systems besser zu analysieren wird das System aufgrund eines Szenarios in die in Abbildung 6 gezeigten vier unterschiedlichen Module, also Termin, Kunden, Bestellung und Bericht aufgeteilt [51].

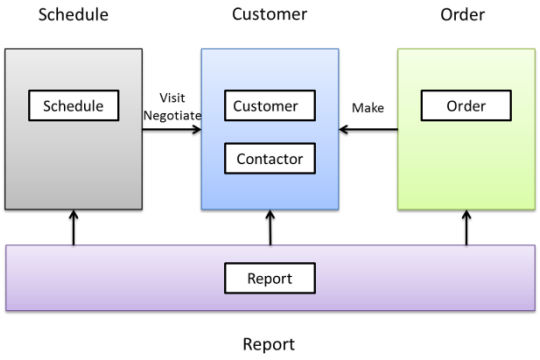

Abbildung 6: Ein Beispiel für den Logical View

Das System wird mithilfe von Struts, Spring und Hibernate implementiert [51], der entsprechende Development View wird in Abbildung 7 gezeigt.

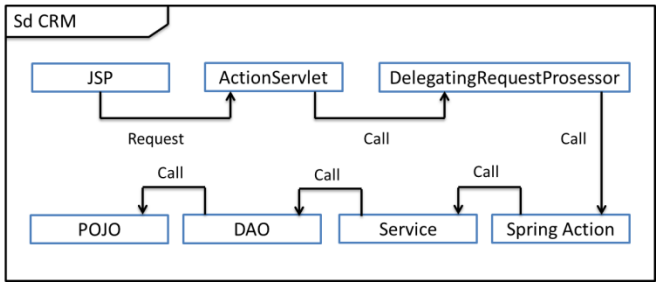

Abbildung 7: Ein Beispiel für den Development View

In Abbildung 8 wird der Physical View dieses Systems gezeigt. Um die mögliche hohe Auslastung zu vermeiden wird das gesamte System getrennt auf drei Servern, also Web Server, Applikations-Server und Datenbank-Server installiert [51].

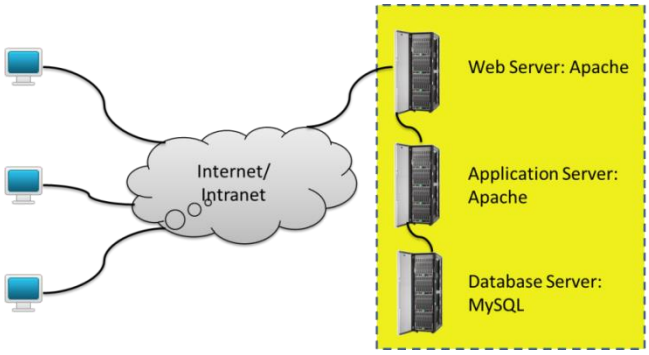

Abbildung 8: Ein Beispiel für den Physical View

Da das 4+1 Sichtenmodell anpassbar ist sowie da es nur einen Prozess gibt, ist der Process View nicht notwendig.

3.3.2 Garland And Anthony's Viewpoints

Das Modell „Garland And Anthony's Viewpoints“ wurde von Jeff Garland und Richard Anthony entwickelt. Die Softwarearchitektursind haben sie „Viewpoints“ genannt, deshalb sind die „Viewpoints“ in diesem Modell eigentlich die Softwarearchitektursichten.

Die Viewpoints dieses Modelles lassen sich in die drei Kategorien, Konzepte und Analysen, logischer Entwurf und Umgebung unterteilen. Die Konzepte und Analysen umfassen die Sichten, die die Problemdomäne beschreiben. Zum logischen Entwurf gehören die Sichten, die die Struktur des zu realisierenden Softwaresystems beschreiben. Die Umgebung enthält z.B. das Deployment. Wie das 4+1 Sichtenmodell können auch

alle Viewpoints mit UML-Diagrammen dokumentiert werden [13]. Im folgendem werden die Viewpoints dieses Modelles kurz vorgestellt.

Folgende Viewpoints gehören zu Konzept und Analyse:

- **„Analysis focused“**. Diese Sicht beschreibt die Entitäten, die für bestimmte Szenarien der Problemdomäne notwendig sind. Deshalb kann diese Sicht auch VOPC [7] benannt werden. Diese Sicht ist üblicherweise als UML Klassendiagramm dokumentiert.

- **„Analysis Interaction“**. Diese Sicht beschäftigt sich mit der Beschreibung der Interaktionen zwischen den Entitäten von „Analysis focused“. Diese Sicht wird normalerweise als UML Interaktionsdiagramm dokumentiert.

- **„Analysis Overall“**. Die Entitäten von „Analysis focused“ werden in dieser Sicht kombiniert, um eine übersichtliche Struktur zu bekommen. Diese Sicht wird wie „Analysis focused“ wieder als UML Klassendiagramm dokumentiert.

- **„Context“**. Die externe Akteure sowie deren Interaktionen mit dem zu realisierenden Softwaresystem werden in dieser Sicht beschrieben. Diese Sicht wird als UML Anwendungsfalldiagramm dokumentiert [13].

Der logische Entwurf umfasst folgende Sichten:

- **„Component“**. Die Softwarekomponenten des zu implementierenden Systems und wie sie miteinander kommunizieren können werden in dieser Sicht als UML Komponentendiagramm dokumentiert.

- **„Component Interaction“**. Diese Sicht beschreibt die Interaktionen zwischen den in der „Component“ Sicht aufgelisteten Softwarekomponenten. Als Ergebnis wird ein UML Interaktionsdiagramm resultiert.

- **„Component State“**. Die Zustände einer Softwarekomponente oder die eng

[7] View Of Participating Classes

damit im Zusammenhang stehenden Softwarekomponenten werden in dieser Sicht beschrieben. Diese Sicht wird normalerweise als UML Zustandsdiagramm oder Aktivitätsdiagramm dokumentiert.

- **„Layered Subsystem“**. Diese Sicht beschreibt die Subsysteme und die Schichten des Softwaresystems. Das UML Paketdiagramm ist eine Möglichkeit für die Dokumentation dieser Sicht.

- **„Logical Data“**. Diese Sicht beschreibt die wichtigen architektonischen Datenstrukturen des Systems. Diese Sicht wird als UML Klassendiagramm dokumentiert.

- **„Subsystem Interface Dependency“**. Diese Sicht beschreibt die Schnittstellen der Subsysteme und die Abhängigkeiten von den Subsystemen. Diese Sicht wird als UML Klassendiagramm dokumentiert [13].

Die Umgebung enthält folgende Sichten:

- **„Deployment“**. Diese Sicht stellt dar, wie die Softwarekomponenten der Hardware zugewiesen werden sollen. Sie wird als UML Verteilungsdiagramm dokumentiert.

- **„Physical Data“**. Die Datenbanken und die dafür notwendige Hardware sowie die Beziehungen zwischen diesen physikalischen Datenbanken sind der Inhalt dieser Sicht. Sie wird auch als UML Verteilungsdiagramm dokumentiert.

- **„Process“**. Die Prozesse einer Softwaresysteminstanz werden in dieser Sicht als UML Verteilungsdiagramm dokumentiert.

- **„Process State“**. Diese Sicht ist wie „Component State“ dynamisch, sie stellt die Zustände jedes Prozesses vor. Das UML Zustandsdiagramm ist für die Dokumentation dieser Sicht geeignet [13].

Es ist leicht zu beobachten, dass dieses Modell sehr viele Sichten umfasst. Dadurch

können nicht nur die Grenze sondern auch die Rolle jeder Sicht genau definiert werden. Trotzdem bringt diese Methode folgende Nachteile: Die Konsistenz zwischen den Sichten ist schwer zu halten; Der Management der Sichten ist schwerer als im 4+1 Sichtenmodell.

3.3.3 Rozanski And Woods's Model

Das Rozanski And Woods's Model wurde von Nick Rozanski und Eoin Woods entwickelt und wurde im Jahr 1995 in ihrem Buch „Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives“ veröffentlicht. Nach ihrer Meinung sollen sieben Sichten bei der Softwarearchitektur analysiert und dokumentiert werden [36]. (vgl. Abbildung 9)

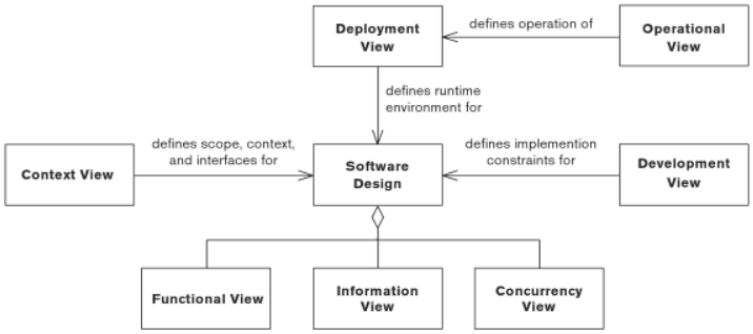

Abbildung 9: Die Sichten von Rozanski And Woods's Model

(Quell: Rozanski, N. und Woods, E., 2011, S. 243)

Die Sichten dieses Modelles sind:

- **Die Kontextsicht**. Die Hauptaufgabe dieser Sicht ist es, die Abhängigkeiten, die Interaktionen und die Beziehungen zwischen dem zu realisierenden Softwaresystem und der Umgebung zu analysieren sowie zu dokumentieren. Unter Umgebung werden die Personen, die Systeme und alle anderen externen Entitäten, mit der das zu realisierende Softwaresystem im Zusammenhang steht, verstanden. Diese Sicht enthält drei Arte von Elementen, das zu realisierende System, die externen Entitäten und die Schnittstellen durch die das System und die externen Entitäten verbunden werden können. Diese Sicht ist normalerweise als UML Kontextdiagramm dokumentiert, aber das

„Boxes-And-Lines“-Diagramm ist auch geeignet.

- **Die funktionelle Sicht**. Die Softwareelemente, die für die Funktionalität des gesamten Softwaresystems verantwortlich sind, sind die Elemente dieser Sicht. Zu analysieren und dokumentiere sind die Funktionalitäten und die Schnittstellen jedes funktionellen Softwareelementes sowie die Interaktionen zwischen solchen Softwareelementen.

- **Die Informationssicht**. Wie die Informationen innerhalb des Softwaresystems gespeichert, manipuliert, verwaltet sowie verteilt sind ist die Forschungsfrage dieser Sicht. Sie beschreibt also die statischen Datenstrukturen und die dynamischen Informationsflüsse.

- **Die Parallelitätssicht**. Diese Sicht beschreibt die parallele Struktur des Systems. Das heißt, jedes funktionelle Softwareelement wird überprüft, um zu entdecken, ob es parallel durchführbar ist. Wie sollen die parallel durchführbaren Softwareelemente koordiniert und kontrolliert soll auch in dieser Sicht berücksichtigt werden. Diese Sicht kann daher die Prozesse, die Threads und die Kommunikation zwischen den Prozessen veranschaulichen.

- **Die Implementierungssicht**. Diese Sicht beschreibt den Prozess der Implementierung. Sie umfasst die Tätigkeiten wie Kodierung, Wartung, Test, Konfiguration, Evaluierung usw.

- **Die Deploymentssicht**. Die Umgebung, mit der das Softwaresystem interagieren wird, wird in dieser Sicht analysiert und dokumentiert. Die Umgebung enthält die für das Softwaresystem notwendig Hardware und die technischen Elemente für die Softwareelemente. Darüber hinaus wird die Allokation der Softwareelemente für die Laufzeitumgebung in dieser Sicht definiert.

- **Die operative Sicht**. Diese Sicht beschäftigt sich damit, die Verwaltung des Systems in der Produktionsumgebung zu beschreiben. Zu den

Verwaltungsarbeiten gehören z.B. Betrieb, Monitoring und Support. [36].

In diesem Abschnitt werden drei Architekturmodelle kurz vorgestellt. Es gibt noch mehr Architekturmodelle, wie The Open Group Architecture Framework[8] (TOGAF), ISO/IEC/IEEE 42010:2011, The Reference Model of Open Distributed Processing[9] (RM-ODP) oder das Architekturmodell von Hofmeister. Trotzdem ist es nicht notwendig, alle möglichen Architekturmodelle zu untersuchen.

3.4 Architekturstile

Es ist nicht schwer zu beobachten, dass einige architektonischen Formen immer wieder verwendet werden, selbst wenn sie für völlig unterschiedliche Softwaresysteme entwickelt wurden. Beispielsweise wird der Architekturstil „Client-Server“ häufig verwendet, wenn man web-basierte Anwendungen entwickeln möchte. Der Architekturstil ist also eine benannte Spezialisierung von Elementen, Beziehungstypen sowie einer Reihe von Constraints. Die Constraints bestimmen wie die Elemente und Beziehungen verwendet können [36,54,55]. Ein Architekturstil ist normalerweise auf bestimmte Softwarearchitektursichten anwendbar. Darüber hinaus stellt er einen Rahmen für die Sicht dar. Das heißt, wenn man sich entscheidet, eine Sicht zu dokumentieren und einen bestimmten Architekturstil zu benutzen, muss man immer nur die im Rahmen dieses Architekturstiles vordefinierten Elemente und Beziehungstypen verwenden und man muss immer die Constraints dieses Architekturstiles beachten [36,55].

Es steht eine Vielzahl an Architekturstilen zur Verfügung. Die Architekturstile lassen sich in die drei Kategorien Modulstil, Allokationsstil und C&C-Stil aufteilen. Der Modulstil definiert eine Menge von Modultypen sowie einige Regeln, die bestimmen, wie die Modultypen kombiniert werden sollen. Der C&C-Stil enthält eine bestimmten Sammlung von Komponententypen und Konnektorentypen. Die möglichen Kombinationen dieser Elementtypen werden durch einige Regeln definiert. Der

[8] www.opengroup.org/togaf/
[9] http://www.rm-odp.net/

Allokationsstil beschreibt, wie die Softwarekomponenten zu den Elementen der Umgebung, in der das Softwaresystem entwickelt oder ausgeführt wird, zugeordnet werden können [7]. In den folgenden drei Abschnitten werden einige wichtigen Architekturstile kurz vorgestellt.

3.4.1 Wichtige Modulstile

Die wichtigen Modulstile sowie deren Elemente, Beziehungen und Constraints werden in Tabelle 1 aufgelistet.

Name	Elementtyp	Beziehungstypen	Constraints
Zersetzungsstil	Modul	Zersetzung	Keine Schleifen. Ein Modul darf nur ein übergeordnetes Modul haben.
Gebrauchsstil	Modul	Gebrauch	Keine.
Verallgemeinerungsstil	Modul	Verallgemeinerung	Keine Schleifen.
Schichtenstil	Schicht	Allowed to use	Jeder Modul befindet sich auf genau einer Schicht. Mindestens zwei Schichten. Keine Schleifen.
Aspekte-Stil	Aspekt	Crosscuts	Keine.
Datenmodell	Datenentität	One-To-One One-To-Many Many-To-Many Generalisierung/Spezialisierung Aggregation	Funktionale Abhängigkeiten sollen vermieden werden.

Tabelle 1: Die wichtigen Modulstile

Der Zersetzungsstil beschreibt die Organisation der Software als Module und Submodule und zeigt wie die Funktionen des Softwaresystems auf diese Module verteilt sind. Da dieser Stil dem „Teile und Herrsche“-Ansatz, der die Komplexität des Softwaresystems senken kann, entspricht, werden fast alle Softwarearchitekturen mit ihm beginnen. Die Beziehungen zwischen den Modulen, also die Zersetzung, sind eine Art von „Is part of“-Beziehungen [7,39,55]. Beispielsweise zeigt Abbildung 10 ein

Diagramm, das den Zersetzungsstil benutzt. Modul A ist der Elternmodul von Modulen B und C, C ist wieder der Elternmodul von D.

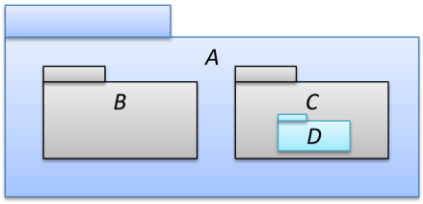

Abbildung 10: Ein Beispiel für den Zersetzungsstil

Während der Zersetzungsstil nur die Organisation der Software als Module und Submodule zeigen kann, kann der Gebrauchsstil auch beschreiben, welche Module welche Module benutzen. Dieser Stil gibt daher an, welche Module existieren müssen, damit das System funktionieren kann. In diesem Sinne ermöglicht dieser Stil die inkrementelle Entwicklung und die Entwicklung einiger nützlichen Subsysteme. Der Beziehungstyp Gebrauch ist tatsächlich eine Art von „Depends on"-Beziehung [7,55]. Abbildung 11 zeigt ein Diagramm, das auf Abbildung 10 aufgebaut ist, als ein Beispiel für den Gebrauchsstil. Im Vergleich zu Abbildung 10 zeigt Abbildung 11 noch, dass Modul B das Modul D benutzt.

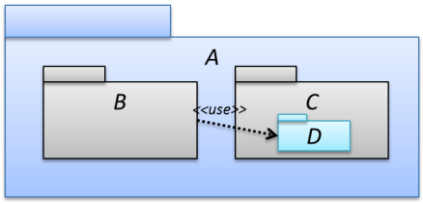

Abbildung 11: Ein Beispiel für den Gebrauchsstil

Ein Verallgemeinerungsstil ergibt sich, wenn die „Is a"-Beziehung eingesetzt wird. Die Module eines solchen Stiles werden so definiert, dass sie Gemeinsamkeiten und die Abweichungen erfassen können. In einer Verallgemeinerung gibt es ein übergeordnetes Modul und ein Kindmodul. Das übergeordnete Modul beschreibt die Gemeinsamkeiten seiner Kinder und wenn es modifiziert wird, werden alle seine Kindermodule automatisch verändert. Die Abweichungen werden bei den Kindermodulen manifestiert. Die Module dieses Stiles können abstrakt sein, das heißt es keine Implementierung notwendig. Darüber hinaus kann ein Kindmodul mehre übergeordnete Module haben, obwohl solche Mehrfachvererbung oft gefährlich ist

[7,55]. Das in Abbildung 12 gezeigte Diagramm gibt ein Beispiel für den Verallgemeinerungsstil. Das Modul Katzen ist das übergeordnete Modul der Module Tiger, Löwen, Leoparden, Luchse, Steppenkatzen usw.

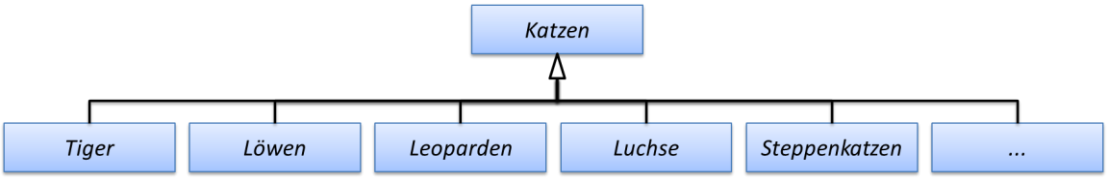

Abbildung 12: Ein Beispiel für den Verallgemeinerungsstil

Der Schichtenstil wird bei der Softwarearchitekturdokumentation am häufigsten verwendet, Beispielsweise verwenden das ISO/OSI-Schichtenmodell, ein Modell für die Kommunikation zwischen Datenschichten in einem offenen heterogenen Netzwerk, und das Security Framework von IBM den Schichtenstil. Unter einer Schicht versteht man eine Gruppe von Modulen, die gemeinsam eine zusammenhängende Reihe von Dienstleistungen für andere Schichten liefern können. Wichtig für die Schicht ist, dass ein Modul sich nur auf einer bestimmten Schicht befinden darf. Ein Modul einer Schicht kann mittels seiner Schnittstelle aufgerufen bzw. abgerufen werden, dies stellt die gemeinsame Funktionalität der Schicht sicher. Die Beziehungen zwischen den Schichten sind „Allowed to use“-Beziehungen, diese Art von Beziehungen ist eine spezielle Form der „Depends on“-Beziehung und ist unidirektional [3,6,7,54].

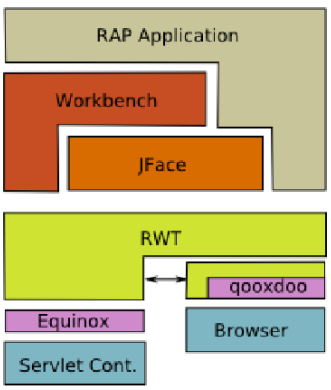

Abbildung 13: Ein Beispiel für den Schichtenstil

(Quell: http://www.educity.cn/wenda/464225.html)

Abbildung 13 zeigt die Architektur der Remote Application Platform (RAP) als ein Beispiel für den Schichtenstil. Wie die Abbildung gezeigt hat enthält diese Architektur

sechs Schichten. Die oberste Schicht ist die RAP Anwendung, die auf Workbench und JFace von Eclipse aufgebaut ist. Dagegen besteht die niedrigste Schicht aus Browser und Servlet Container die nur von anderen Schichten benutzt werden können.

Der Aspekte-Stil wird verwendet um die Module nach ihrer Verantwortung hinsichtlich der übergreifenden Aspekte zu isolieren. Dieser Stil ist für die Aspektorientierte Programmierung (AOP) von Bedeutung. Die Elemente dieses Stiles sind Aspekte, die Beziehungen dazwischen heißen „Crosscut"-Beziehungen, die besagen dass die generische Funktionalitäten eines Aspektes von einem anderen Aspekt abhängig sind oder durch einen anderen Aspekt beeinflusst werden können. Ein Aspekt kann andere Aspekte, andere normale Module oder sich „crosscutten" [7]. Es wird angenommen, dass ein Online-Shop-System aus den Modulen Authentifizierung, Datenbankzugriff und Artikelverwaltung besteht. Das Modul Artikelverwaltung kann nur den aktuellen Lagerbestand ermitteln. Da das Modul Datenbankzugriff für die Funktionalität von des Moduls Artikelverwaltung dringend notwendig ist, können diese beiden Module gemeinsam betrachtet werden. Ein solcher Aspekt ist dann ein übergreifender Aspekt und das Modul Datenbankzugriff ist ein „Crosscutting Concern" der Artikelverwaltung. Abbildung 14 zeigt die Struktur der Anwendung „IkeWiki" als ein weiteres Beispiel. Es gibt vier Aspekte in diesem Diagramm und drei davon besitzen „Crosscut"-Beziehungen.

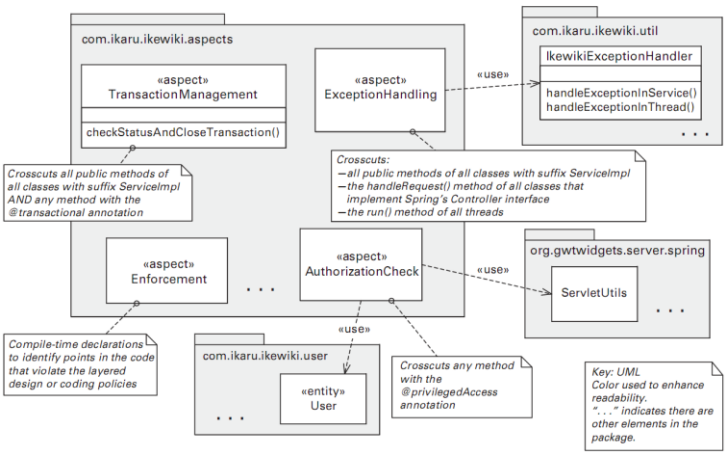

Abbildung 14: Ein Beispiel für den Aspekte-Stil

(Quell: Clements P. et al., 2011, S. 107)

Die Datenmodellierung ist eine häufige Aktivität im Softwareentwicklungsprozess und resultiert in einem Datenmodell. Das Datenmodell beschreibt die statischen Informationen in Bezug auf die Datenentitäten und deren Beziehungen. Die Elemente dieses Stiles sind die Datenentitäten. Unter einer Datenentität versteht man ein unterscheidbares Objekt, das die Informationen, die im Softwaresystem gespeichert oder repräsentiert werden, enthält. Eine Datenentität kann unterschiedliche Eigenschaften wie Name, Datenattribute usw. haben. Die Beziehungen zwischen den Datenentitäten lassen sich in drei Kategorien, also Generalisierung/Spezialisierung, Aggregation und die Beziehung die die logische Assoziation der Datenentitäten, also One-To-One, One-To-Many und Many-To-Many, darstellt, unterteilen [7,37,55]. Untenstehende Abbildung zeigt ein Datenmodell. Es gibt fünf Datenentitäten im diesem Datenmodell. InformationObject ist die generalisierte Datenentität von Person, Content und Organisation, das heißt z.B. zwischen InformationObject und Person die „Generalisierung/Spezialisierung“-Beziehung. Darüber hinaus gibt es zum Beispiel zwischen Person und Extension eine „One-To-Many“-Beziehung.

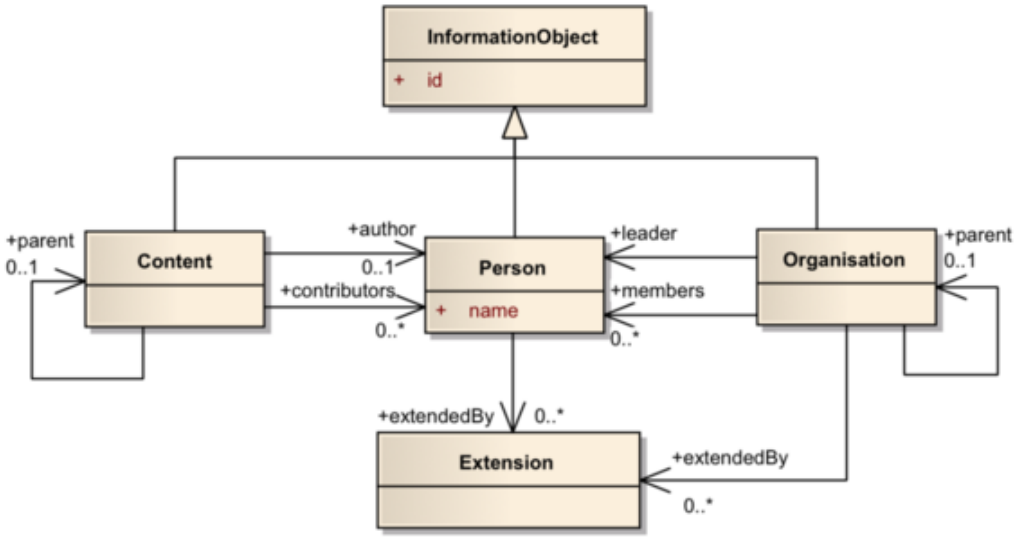

Abbildung 15: Ein Beispiel für das Datenmodell

(Quell: http://www.soziotech.org/communitymashup/model-driven-approach/)

3.4.2 Wichtige C&C-Stile

Die wichtigen C&C-Stile sowie deren Komponenten und Konnektoren werden in untenstehender Tabelle aufgelistet.

Name	Komponente	Konnektor
„Pipe-And-Filter“-Stil	Filter	Rohr
Client-Server-Stil	Client Server	Anfrage/Antwort-Konnektor
„Peer-To-Peer“-Stil	Peer	Anruf-Rückkehr-Konnektor
SOA-Stil	Dienstanbieter Dienstnehmer ESB Serviceregistrierung Orchestrierungsserver	Anruf-Rückkehr-Konnektor Messaging-Konnektor
„Pub/Sub“-Stil	C&C Komponente mit Publish-/Subscribe-Ports	Publish-Subscribe-Konnektor
„Shared-Data“-Stil	Component-Repository Datenzugriff-Komponente	Data-Konnektor

Tabelle 2: Die wichtigen C&C-Stile

Der „Pipe-And-Filter“-Stil wird durch die aufeinanderfolgenden Transformationen von Datenströmen gekennzeichnet. Die Komponenten dieses Stiles heißen Filter. Ein Filter transformiert die in seinen eingehenden Port einfließenden Informationen und sendet die transformierten Informationen über seinen ausgehenden Port. Ein Filter muss mit der zu transformierenden Information kompatibel sein. Mehre Filter können in der Regel gleichzeitig oder schrittweise ausgeführt werden. Ein Konnektor dieses Stiles heißt Rohr. Der eingehende Port eines Filters und der ausgehende Port eines anderen Filters werden durch ein Rohr verbunden, deshalb stellt ein Rohr den Kanal für die Datenübertragung dar, und das Rohr verändert die zu übertragenden Daten nicht [7,36,54,55]. Abbildung 16 zeigt ein Diagramm, das den „Pipe-And-Filter“-Stil benutzt. Es wurde mit der webbasierten Anwendung Yahoo! Pipes[10] gemacht. Im diesem Diagram gibt es zwei Filter, die durch ein Rohr verbunden sind.

[10] https://pipes.yahoo.com/pipes/

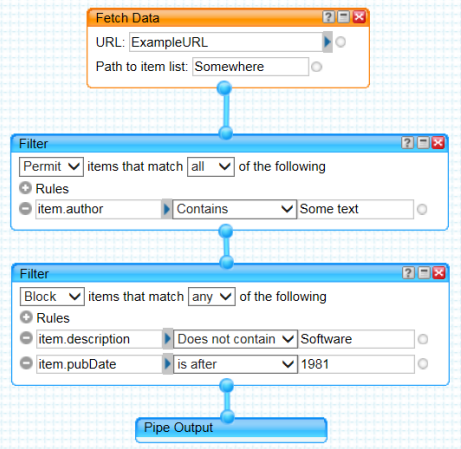

Abbildung 16: Ein Beispiel für den „Pipe-And-Filter“-Stil

Die Komponenten von Client-Server-Stil sind Clients und Server. Ein Server kann bestimmte Dienstleistungen, die eine unbestimmte Anzahl von Clients benutzen können, liefern. Einige Komponenten dieses Stiles können gleichzeitig Server und Client sein, dies ist dann der Fall, wenn z.B. eine Komponente die Rolle eines Intermediär spielt. Ein Client und ein Server werden durch einen Anfrage/Antwort-Konnektor verbunden [3,7,55].

Abbildung 17 zeigt die Client-Server-Struktur von WebVbMF. Die Komponente „Main Server“ ist gleichzeitig Server und Client, da sie nicht nur die Dienstleistungen von „Database“ benutzt sondern auch den Clients Dienstleistungen liefern.

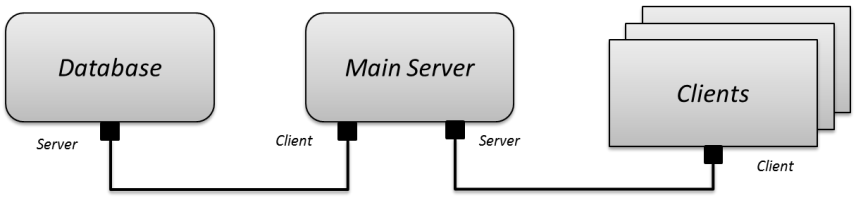

Abbildung 17: Ein Beispiel für den Client-Server-Stil

Der „Peer-To-Peer“-Stil ist wie der Client-Server-Stil ein Anfrage-Antwort-Stil. Im Unterschied zu dem Client-Server-Stil gibt es im „Peer-To-Peer“-Stil keine Clients und Server, sondern „Peers“. Das heißt, dass jeder „Peer“ gleichzeitig die Rolle Client und Server hat. Ein „Peer“ kann daher im Prinzip mit jeder anderen „Peer“ interagieren. Jede „Peer“ muss bestimmte Dienstleistungen, die alle anderen „Peers“ benutzen können, liefern. Beispiele für die „Peer-To-Peer“-Systeme sind BitTorrent, Skype usw. Zwei

„Peers“ werden durch einen Anfrage-Antwort-Konnektor verbunden [7,55].

Abbildung 18 zeigt ein Beispiel für den „Peer-To-Peer“-Stil. Es gibt drei „Peers“ in diesem Diagramm und jeder „Peer“ kann mit jedem anderen „Peer“ interagieren.

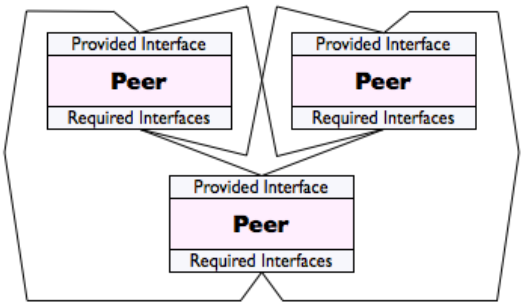

Abbildung 18: Ein Beispiel für den „Peer-To-Peer“-Stil

(Quell: http://www.thomasalspaugh.org/pub/fnd/architecture.html)

Der SOA-Stil ist eine andere Art Anfrage-Antwort-Stil. SOA ist die Abkürzung für „Service-Oriented Architecture“ (Deutsch: Serviceorientierte Architektur). Die SOA besteht aus einer Vielzahl von verteilten Komponenten, die Dienstleistungen bereitstellen und/oder nutzen können. Die Komponente, die die Dienstleistungen anderen Komponenten benutzen, heißen Dienstnehmer, und die Komponenten, die die Dienstleistung liefern, heißen Dienstanbieter. Manchmal kann eine Komponente nicht nur Diensteanbieter sondern auch Dienstnehmer sein. In der SOA können die Dienstanbieter und Dienstnehmer in der Regel getrennt, von unterschiedlichen Organisationen für unterschiedliche Systeme entwickelt werden. Darüber hinaus können sie in unterschiedlichen Programmiersprachen geschrieben sowie auf unterschiedlichen Plattformen ausgeführt werden. Deshalb sind in einer SOA einige zusätzliche Komponenten notwendig, um die Zusammenarbeit von Dienstnehmern und Dienstanbietern sicherzustellen. Diese zusätzlichen Komponenten sind z.B. ESB (Enterprise Service Bus, ein Zwischenelement für das Routing und die Transformation von Nachrichten zwischen Dienstanbietern und Dienstnehmern.), Serviceregistrierung (Wird von einem Dienstanbieter benutzt, um seine Dienstleistungen zu registrieren sowie von einem Dienstnehmer benutzt, um die Dienstleistungen zu entdecken.) und Orchestrierungsserver (Wird benutzt, um anhand von vordefinierten Skripten die

Interaktionen zwischen Dienstanbietern und Dienstnehmern zu koordinieren.). Die Dienstanbieter und Dienstnehmer können durch einen Messaging-Konnektor oder Anfrage-Antwort-Konnektor verbunden werden. Ein Messaging-Konnektor wird verwendet, um den asynchronen Nachrichtenaustausch sicherzustellen. Anhand des verwendeten Protokolls kann ein Anfrage-Antwort-Konnektor SOAP-Konnektor oder REST-Konnektor sein [7,39,55]. Die untenstehende Abbildung zeigt die Struktur von Adventure Builder[11], die den SOA-Stil verwendet. OPC steht für „Order Processing Center“ und liefert unterschiedliche Dienstleistungen an die Komponenten „Consumer Website“, „Airline Provider“, „Lodging Provider“ und „Activity Provider“. Gleichzeitig benutzt OPC die Dienstleistungen von „Bank“, „Airline Provider“, „Lodging Provider“ und „Activity Provider“.

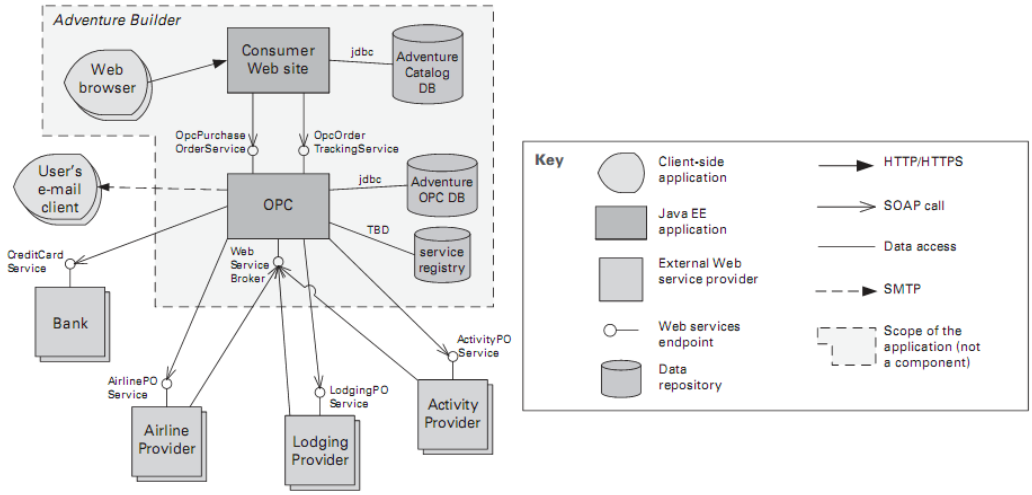

Abbildung 19: Ein Beispiel für den SOA-Stil

(Quell: Clements P. et al., 2011, S. 173)

In den „Pub/Sub“-Stil (Publish-Subscribe Style) interagieren die Komponenten über angekündigte Ereignisse. Es ist die Aufgabe der Infrastruktur, sicherzustellen, dass alle Komponenten, die ein bestimmtes Ereignis abonniert haben, dieses Ereignis auch bekommen können. Die Komponenten dieses Stiles können jede Art von C&C Komponenten, die zumindest einen Publish-Port oder einen Subscribe-Port haben, sein. Die Komponenten dieses Stiles werden durch die sogenannten

[11] wiki.sei.cmu.edu/sad/index.php/Main_Page

Publish-Subscribe-Konnektoren verbunden. Diese Konnektorart hat die Aufgabe die Ereignisse innerhalb des Systems zu verteilen [7,55].

Die Elemente des „Shared-Data“-Stiles interagieren miteinander durch den Austausch von persistenten Daten. Die Komponente die die Daten persistent speichert heißt Component-Repository. Auf die in einem Component-Repository gespeicherten Daten kann über unterschiedliche Methoden zugegriffen werden. Die Komponenten, die die Zugriffsmethoden realisieren, heißen Datenzugriff-Komponenten. Die Component-Repositorys und Datenzugriff-Komponenten werden durch die sogenannten Data-Konnektoren verbunden. Ein Data-Konnektor bezeichnet, welche Datenzugriffs-Komponente die Daten in welchem Component-Repository lesen und/oder schreiben darf [7].

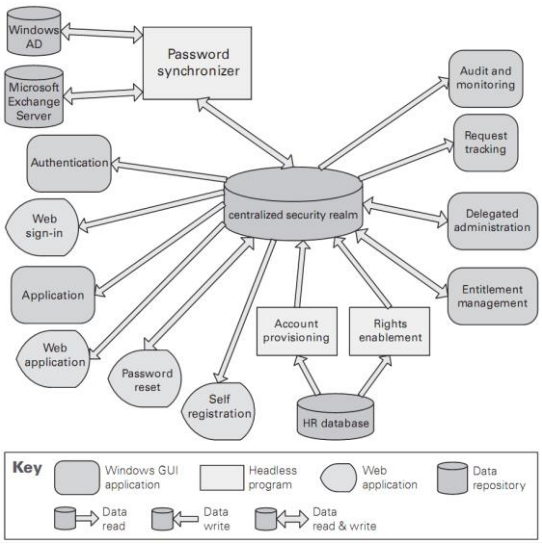

Abbildung 20: Ein Beispiel für den „Shared-Data“-Stil

(Quell: Clements P. et al., 2011, S. 181)

Abbildung 20 zeigt ein Beispiel für diesen Stil. Es gibt insgesamt drei Component-Repositorys und mehre Datenzugriffs-Komponenten. Die Komponente „Centralized Security Realm“ ist das Haupt-Component-Repository, viele Komponenten können darauf zugreifen. Die Datenzugriffs-Komponenten besitzen unterschiedliche Rechte für den „Centralized Security Realm“, z.B. „Password Synchoronizer“ kann die Daten im „Centralized Security Realm“ lesen und schreiben, „Password Reset“ kann die Daten schreiben und „Application“ kann nur die Daten lesen.

3.4.3 Wichtige Allokationsstile

Die wichtigen Allokationsstile sowie deren Elementtypen und Beziehungstypen werden in untenstehender Tabelle aufgelistet.

Name	Elementtypen	Beziehungstypen
Deployment-Stil	Softwareelement Umgebungselement	Allocated-to Migrates-to Copy-migrates-to Execution-migrates-to
Installationsstil	Softwareelement Umgebungselement	Allocated-to Containment
Arbeitszuweisungsstil	Softwareelement Umgebungselement	Allocated-to

Tabelle 3: Die wichtigen Allokationsstile

Der Deployment-Stil beschäftigt sich damit, die Softwareelemente eines Systems den Umgebungselementen zuzuweisen. Unter einem Softwareelement wird hier z.B. eine Komponente des Component-And-Connector Stils verstanden. Ein Umgebungselement ist die Hardware der Rechnerplattform, auf dem sich das Softwaresystem befindet oder ausgeführt wird. Der Deployment-Stil kann daher sicherstellen, dass die Anforderungen der Softwareelemente an der Umgebung erfüllen können. Die Beziehungen zwischen einem Softwareelement und einem Umgebungselement ist grundsätzlich „Allocated-to“, dies besagt auf welchem Umgebungselement sich ein Softwareelement befindet oder ausgeführt wird. Die Allokation kann auch dynamisch sein, das heißt, dass sich die Allokation während der Laufzeit ändern kann. Die dynamischen Beziehungen dieses Stiles sind „Migrates-to“, „Copy-migrates-to“ und „Execution-migrates-to“ [7,39]. Nehmen wir an, dass das Softwareelement A dem Umgebungselement B zugewiesen ist. „Migrates-to“ bedeutet, dass A sich während der Laufzeit nicht mehr auf B, sondern auf einem anderen Umgebungselement C befindet und dort ausgeführt wird. „Copy-migrates-to“ besagt, dass A sich gleichzeitig auf B und C befindet, aber nur auf C ausgeführt wird. Betrachten wir ein anderes Beispiel für „Execution-migrates-to“: Softwareelement A befindet sich gleichzeitig auf B und C. A wird auf einem aktivierten

Umgebungselement ausgeführt, das heißt, dass A auf B oder C ausgeführt werden kann, aber nicht gleichzeitig auf B und C ausgeführt werden darf. Die untenstehende Abbildung zeigt, das Softwareelement „EnterpriseWebApp.ear“ das dem Umgebungselement „Application Server“ und „app-client.jar“ das dem „Admin user PC“ zugewiesen ist.

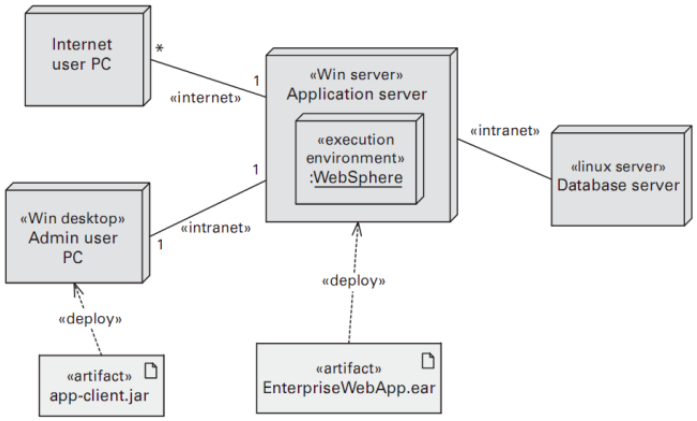

Abbildung 21: Ein Beispiel für den Deployment-Stil

(Quell: Clements P. et al., 2011, S. 198)

Sobald das gewünschte Softwaresystem implementiert ist, müssen die resultierenden Dateien wie Bibliotheken, ausführbare Dateien, Konfigurationsdateien, Log-Dateien, Hilfedateien, Lizenzdateien, Versionskontrolldateien usw. verpackt werden, sodass sie auf der Zielplattform installiert werden können. Der Installationsstil dient zu Organisierung solcher Dateien, er beschreibt daher die Zuordnung der Softwarekomponenten der Softwarearchitektur in dem Dateisystem in der Produktionsumgebung. Die Elemente dieses Stiles sind Softwareelemente und Umgebungselemente. Ein Umgebungselement in diesem Sinne ist eine Datei oder ein Ordner. Die Beziehungen in diesem Stil sind „Allocated-to“ und „Containment“. „Allocated-to“ wird verwendet, um die Beziehung zwischen einem Softwareelement und einem Umgebungselement zu bestimmen, also welches Softwareelement zu welchem Umgebungselement zugewiesen werden muss. „Containment“ bestimmt die Beziehungen zwischen zwei Umgebungselementen, also welches Umgebungselement welchem Umgebungselement übergeordnet werden muss, diese Art von Beziehungen

erzeugt eine baumartige Struktur [7]. Folgende Abbildung zeigt ein Diagramm, das den Installationsstil benutzt. Die Softwareelemente „Dispatcher“ und *.jsp“ sind in der Umgebungselement „web-client.war“ verpackt. „web-client.war“ enthält zwei andere Umgebungselemente „Struts.jar“ und „WebMessages.properties“.

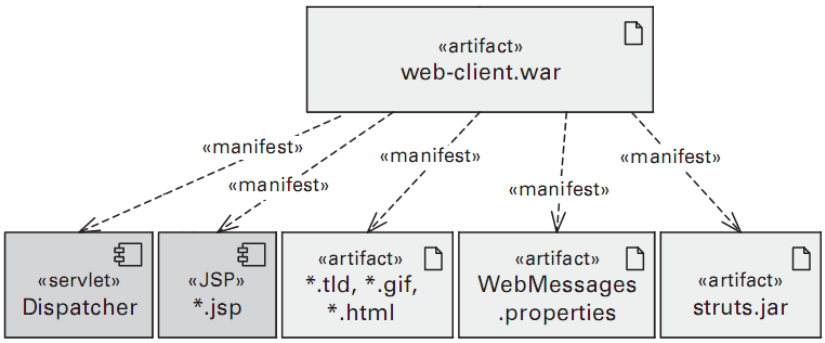

Abbildung 22: Ein Beispiel für den Installationsstil

Der Arbeitszuweisungsstil, wie es der Name schon sagt, beschäftigt sich damit, die Softwareelemente Teams und/oder Personen zuzuweisen. Jedes Softwareelement wird dann von dem/der durch den Arbeitszuweisungsstil festgelegtem Team/Person realisiert. Dieser Stil umfasst nicht nur die Kodierung, sondern auch viele andere Aufgaben während der Implementierung wie Tests, Konfiguration, Evaluierung usw. Die Elemente dieses Stiles sind daher Softwareelemente und Umgebungselemente. Das Umgebungselement in diesem Stil soll als Team oder Person verstanden werden. Die Beziehung zwischen diesen zwei Arten Elementen heißt „Allocated-to“ und besagt, dass ein Team oder eine Person die Realisierung eines Softwareelementes übernimmt [7].

3.5 Weg zur graphischen Dokumentation

Mit dem in den Abschnitten 3.2 bis 3.4 vorgestellten Wissen kann der Prozess zur graphischen Dokumentation der Softwarearchitektur jetzt begonnen werden.

Da die Natur der Softwarearchitekturdokumentation die Dokumentation relevanter Softwarearchitektursichten ist, ist der erste Schritt zur Softwarearchitekturdokumentation zu bestimmen, welche Sichten dokumentiert werden sollen. Die im Abschnitt 3.3 vorgestellten Architekturmodelle können dazu beitragen, die wichtigen Sichten

herauszufinden. Deshalb können die zu dokumentierenden Sichten in diesem Schritt mithilfe von Architekturmodellen bestimmt werden.

Wenn die durch die Architekturmodelle entdeckten Sichten für die Softwarearchitekturdokumentation nicht geeignet sind, kann man das Ergebnis modifizieren, indem man einige Sichten entfernt oder einige Sichten hinzunimmt. Oder man kann die Sicht wie folgt selbst bestimmen.

Zur Bestimmung der zu dokumentierenden Sichten muss man zuerst herausfinden, welche Stakeholder die Softwarearchitekturdokumentation benötigen und was das Interesse jedes Stakeholders an der Dokumentation ist. Folgende Tabelle listet die möglichen Gruppen von Stakeholdern und ihr Interesse an unterschiedlichen Arten von Sichten auf. Angesichts der Organisationsformen und der Kontexte der Projekte können unterschiedliche Gruppen von Stakeholdern involviert werden, deshalb müssen nicht alle in dieser Tabelle gelisteten Stakeholder berücksichtigt werden [3,7].

Stakeholder	**Interesse an Modulsichten**	**Interesse an C&C-Sichten**	**Interesse an Allokationssichten**
Projektleiter	Gering	Kein	Groß
Implementierungsteam	Groß	Groß	Normal
Tester und Systemintegrierer	Groß	Normal	Groß
Designer anderen Systeme	Groß	Gering	Kein
Wartungsarbeiter	Groß	Groß	Normal
Anwendungsentwickler	Groß	Gering	Groß
Kunden	Gering	Gering	Groß
Endbenutzer	Kein	Gering	Gering
Analysten	Groß	Groß	Groß
Infrastrukturbetreiber	Gering	Normal	Normal
Architekten	Groß	Groß	Groß

Tabelle 4: Die Stakeholder und ihr Interesse an den verschiedenen Sichten

(Quell: Clements P. et al., 2011, S. 316-325)

Wenn ein Stakeholder großes Interesse an einer Sicht hat, soll diese Sicht so dokumentiert werden, dass die resultierende Dokumentation die Informationen über diese Sicht detailliert wiederspiegeln kann. Wenn das Interesse normal ist, sollen nur

manche Informationen über die Sicht detailliert dokumentiert werden. Für geringes Interesse müssen nur die Überblicksinformationen dokumentiert werden. Wenn kein Stakeholder Interesse an einer Sicht hat, soll sie nicht dokumentiert werden [3,7].

Im Folgendem wird der Stakeholder Projektleiter als ein Beispiel kurz vorgestellt. Ein Projektleiter soll sich um den Zeitplan, die verfügbaren Ressourcen usw. kümmern. Um solche Arbeiten zu erledigen braucht er Informationen über die zu implementierenden Module, die Komplexität jedes Modules sowie die Abhängigkeiten zwischen den Modulen. Er hat kein Interesse an den technischen Informationen wie der Interaktionen zwischen den Softwareelementen. Deshalb sind für den Projektleiter grundsätzlich nur die Allokationssichten wichtig [7]. Die Analysen anderen Stakeholder sind ähnlich und können daher unterlassen werden.

Anhand obiger Tabelle kann durch die Aufzählung der existierenden Stakeholder der Softwarearchitektur eine Sammlung von Sichten gefunden werden. Die Menge dieser Sichten ist normalerweise groß. Um eine besser handhabbare Menge von Sichten zu bekommen müssen manche Sichten entfernt werden. Zu entfernenede Sichten sind z.B. die Sichten, die nur sehr wenige Stakeholder unterstützen sowie die Sichten, die durch andere Sichten ersetzt werden können [3,7].

Wenn die zu dokumentierenden Sichten gefunden sind, müssen sie jetzt graphisch dokumentiert werden. Wie eine Sicht graphisch dokumentiert werden soll hängt von dem Kontext der Sicht ab. Das heißt, man muss vor der Dokumentation zuerst die Sicht analysieren und Antworten auf folgende Fragen finden:

- Weist diese Sicht einen bestimmten Stil auf?
- Welches Diagramm ist für die Dokumentation dieser Sicht geeignet?
- Welches graphische Architektur-Tool muss für die Dokumentierung dieser Sicht verwendet werden?

Wenn eine Sicht einen bestimmten Stil aufweist, können die in diesem Stil definierten

Elementtypen, Beziehungstypen und Constraints verwendet werden. Andernfalls muss man die Elementtypen, Beziehungstypen und Constraints selbst definieren. Danach kann anhand der Elementtypen, Beziehungstypen und Constraints eine Diagrammart für die graphische Dokumentation auswählen. Am Ende kann man mithilfe von einem Architektur-Tool, das die gewählte Art von Diagramm erzeugen kann, die Sicht graphisch dokumentiert werden. Wenn alle gewählten Sicht graphisch dokumentiert sind, bekommt man die graphische Softwarearchitekturdokumentation.

Die Wahl des geeigneten Stiles hängt von der Natur der Ablaufstrukturen des Softwaresystems und dem Verwendungszweck der Dokumente der Softwarearchitektur ab. Beispielsweise, wenn das System auf eine Vielzahl von Datenbank zugreifen muss, ist wahrscheinlich ein „Shared-Data“-Stil geeignet. Wenn die hohe Leistung für das zu entwickelte System wichtig ist, muss man einen solchen Stil wählen, der die Leistungsanalyse ermöglicht.

4 Graphische Architektur-Tools

Die möglichen Softwarearchitekturdokumentationsformen wurden bereits im Abschnitt 3.1 vorgestellt. Eine Softwarearchitektur kann graphisch mittels Bildern oder Diagrammen dokumentiert werden. Da die Bilder nicht universelle Eigenschaften besitzen, werden in diesem Kapitel nur Architektur-Tools, die die Softwarearchitektur als Diagramme dokumentieren können, vorgestellt.

4.1 UML-basierte Anwendungen

Die am weitesten verbreitete Methode zur Softwarearchitekturdokumentation ist die UML, sie ist eine grafische Modellierungssprache und wurde im Jahr 1995 von Grady Booch, Ivar Jacobson und James Rumbaugh entwickelt. Die UML wurde von der Object Management Group (OMG) und der ISO standardisiert und daher ist die dominierende Sprache für die Modellierung [34].

Die UML definiert insgesamt vierzehn unterschiedliche Diagrammarten, die sich in die zwei Kategorien Strukturdiagramme und Verhaltensdiagramme unterteilen lassen. Zu den Strukturdiagrammen gehören das Klassendiagramm, das Kompositionsstrukturdiagramm, das Komponentendiagramm, das Verteilungsdiagramm, das Objektdiagramm, das Paketdiagramm und das Profildiagramm. Die Verhaltensdiagramme können die dynamischen Eigenschaften des Systems beschreiben und umfassen das Aktivitätsdiagramm, das Anwendungsfalldiagramm, das Interaktionsübersichtsdiagramm, das Kommunikationsdiagramm, das Sequenzdiagramm, das Zeitverlaufsdiagramm und das Zustandsdiagramm [34].

Nach der Entstehung von UML wurden zahlreiche Anwendungen für die UML entwickelt. Man kann heute UML Diagramme nicht nur auf Computern mit den Betriebssystemen Windows, Linux, Unix, Solaris oder MacOS sondern auch auf mobilen Plattformen wie Smartphones und Tablets mit dem Betriebssystemen iOS oder Android bearbeiten. Die Anwendungen für Personal Computer oder Server sind z.B.

Enterprise Architect von Sparx Systems[12], BOUML[13], Rational Software Architect[14] (RSA) von IBM usw. Zu den Anwendungen für mobile Plattformen zählen z.B. AndyUML[15] von yUML, Cubetto UML[16] von Semture GmbH usw. Darüber hinaus ist die Verwendung eines browserbasierten Editors, wie Cacco[17], Lucidchart[18], GenMyModel[19] usw. eine andere Variante von der Bearbeitung von UML Diagrammen [47].

Obwohl die UML sehr weit verbreitet ist, ist sie für die graphische Dokumentation der Softwarearchitektur nur bedingt anwendbar, da die von der UML definierten vierzehn Arten von Diagrammen die für die Softwarearchitekturdokumentation notwendigen Diagrammarten nicht abdecken können. Beispielsweise gibt es kein eingebautes Diagramm für den Schichtenstil in der UML. Trotzdem sind manche UML-basierten Anwendungen wie die RSA grundsätzlich für die graphische Softwarearchitekturdokumenation geeignet, da sie zusätzliche Elementtypen eingeführt haben, sodass mehre Arten von Diagrammen erzeugt werden können. Im Folgenden wird die RSA als ein Beispiel UML-basierter Anwendungen vorgestellt.

Die RSA wurde primär von Rational Software entwickelt, nach der Übernahme der Rational Software durch IBM im Jahr 2003 wurde sie von IBM weiter entwickelt und aktualisiert. Die neuste Version der RSA ist 9.1.1. [33].

Die RSA basiert auf dem quelloffenen Eclipse-Software-Framework und kann durch zahlreiche Eclipse-Plugins erweitert werden [16]. Deshalb kann die RSA die Zusammenarbeit von Architekten, die gewöhnlich Eclipse als IDE verwenden, erleichtern. Die RSA unterstützt auch die automatische Codegenerierung in Java, C#, C++, Enterprise JavaBean (EJB), Web Services Description Language (WSDL), XML Schema Definition (XSD) und CORBA Interface Description Language (IDL).

[12] http://www.sparxsystems.com/products/ea/index.html
[13] http://www.bouml.fr/
[14] http://www.ibm.com/developerworks/downloads/r/architect/
[15] http://yuml.me/
[16] http://cubetto.semture.de/
[17] https://cacoo.com
[18] https://www.lucidchart.com/
[19] http://www.genmymodel.com/

Umgekehrt kann die RSA Code in Java, C++ oder .NET zur UML Diagrammen transformieren [33].

Der Begriff „Template“ ist in der RSA wichtig, mithilfe eines Templates kann ein Modell, das nur die von diesem Template erlaubten Diagrammarten enthalten darf, erzeugt werden, beispielsweise wenn man unter Verwendung der Template „Rose Package“ ein Modell erzeugt, kann man dann unterschiedliche UML-Diagramme definieren. Manche Templates sind speziell für den Entwurf und die Softwarearchitekturdokumentation eingeführt worden, sie werden in Abbildung 23 gezeigt [32].

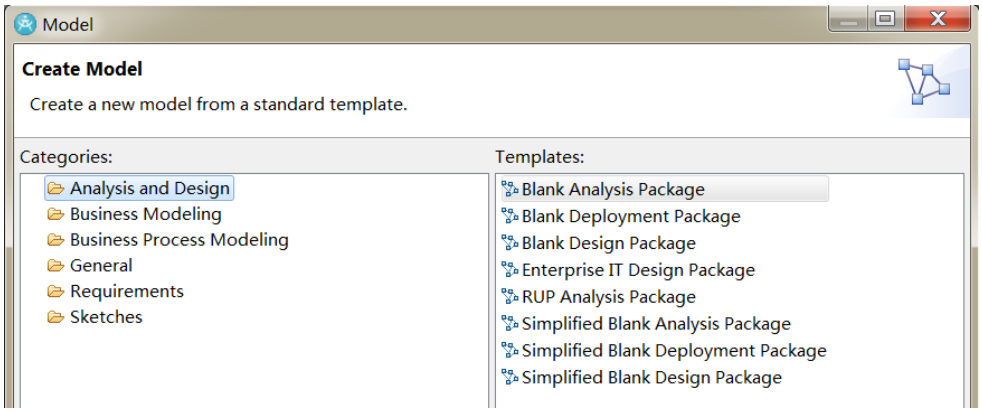

Abbildung 23: Die Templates im RSA

Diese Templates entsprechen teilweise dem 4+1 Sichtenmodell. Die Templates „Analysis Package“ und „Design Package“ können verwendet werden, um den Logical View sowie den Process View zu dokumentieren. Anwendungsfalldiagramme, Aktivitätsdiagramme, Klassendiagramme, Komponentendiagramme, Sequenzdiagramme sind in diesen Templates erlaubt. Mithilfe des Template „Deployment Package“ kann der Deployment View als Verteilungsdiagramm dokumentiert werden. Um das Szenario zu dokumentieren kann das Template „Use Case Package“ benutzt werden [32].

Der Physical View kann in der RSA als „Topology“ dokumentiert werden. Eine Topology ist kein UML-Diagramm und deshalb nicht universal. Folgende Abbildung zeigt ein mit RSA erzeugte Topology. Dieses Diagramm beschreibt die Architektur einer Anwendung und kann als das Dokumentation der Physical View dieser Anwendung

verwendet werden [32].

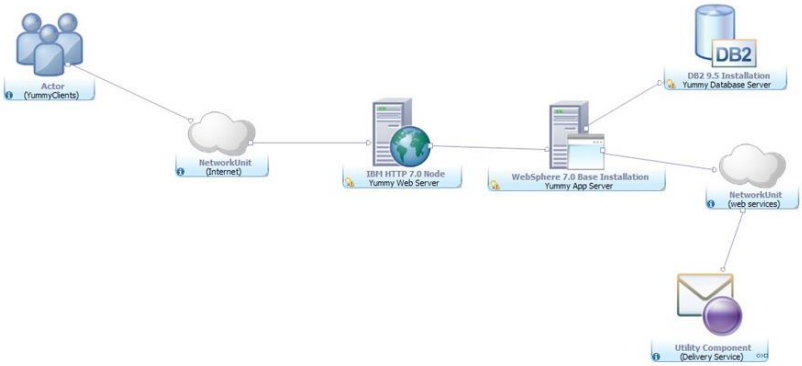

Abbildung 24: Die Topologie einer Anwendung

(Quell: http://www.ibm.com/developerworks/cn/rational/define-application-architecture-rational-software-architect-1/index.html)

Es ist auch möglich, mit RSA sogenannte „Sketches“ zu erzeugen. Bei der Softwarearchitekturdokumentation kann der Sketch verwendet werden, um alle möglichen Sichten graphisch zu beschreiben. Die Sichten, die schwer als UML-Diagramme zu dokumentieren sind, können mithilfe von Sketches dokumentiert werden. Wie schon gesprochen hat ist der Schichtenstil schwer als UML-Diagramm zu beschreiben, der in der folgenden Abbildung gezeigte Sketch kann aber solche Stil dokumentieren. Dieser Sketch beschreibt den Logical View einer Anwendung, es ist auch möglich, andere Views als Sketches zu dokumentieren.

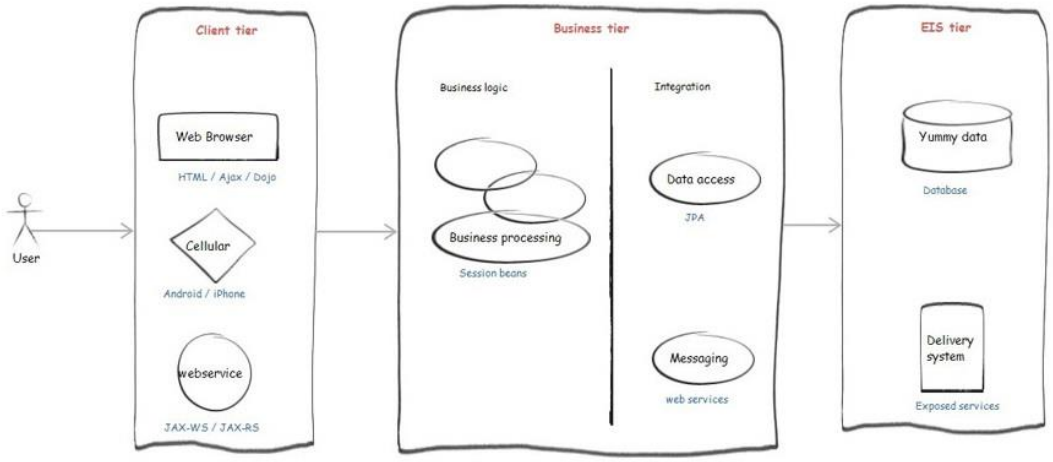

Abbildung 25: Ein Sketch

4.2 yEd Graph Editor

Der yEd Graph Editor wurde von yWorks GmbH entwickelt. Es ist eine kostenlose Desktop-Applikation und kann auf unterschiedlichen Plattformen wie Windows, Unix, Linux und Mac OS X laufen [52].

Der yEd Graph Editor unterscheidet nicht zwischen verschiedenen Diagrammarten. Das heißt, es gibt keine Klassendiagramme, Zustandsdiagramme usw., sondern nur Diagramme. Die Elementtypen, die Beziehungstypen und die Constraints können von dem Benutzer frei gewählt werden [52]. Folgende Abbildung zeigt alle möglichen Elementtypen des yEd Graph Editor.

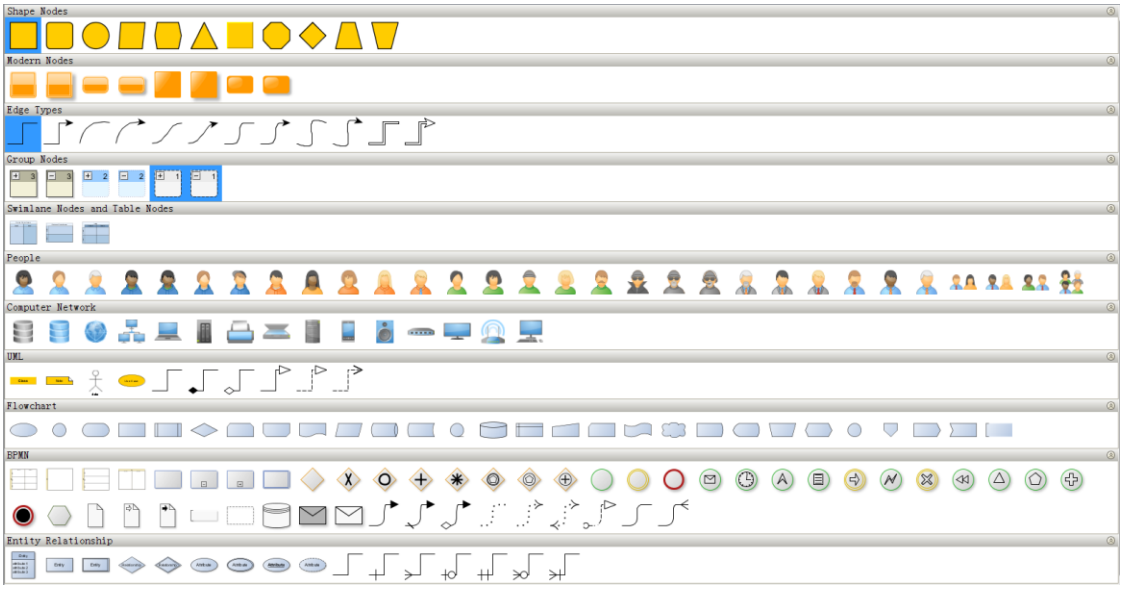

Abbildung 26: Die möglichen Elementtypen in dem yEd Graph Editor

Wenn die durch den yEd Graph Editor vordefinierten Elementtypen für ein Diagramm nicht tauglich sind, können externe Elementtypen eingeführt werden [52]. Deshalb kann der yEd Graph Editor theoretisch alle Arten von Diagrammen erzeugen. Die üblichen mithilfe dieser Anwendung erzeugten Diagramme sind aber UML Diagramme, Business Process Modeling Notation (BPMN)-Diagramme, Flussdiagramme usw. Trotz der unbegrenzten Erweiterbarkeit macht diese Eigenschaft den yEd Graph Editor nicht standardisiert, das heißt, dass die selbst definierten Elementtypen den Verwendungsumfang der Diagramme beschränken können. Die Anwendung kann daher

dann benutzt, wenn andere Anwendungen die gewünschten Diagramme nicht oder nicht effizient erzeugen können sowie wenn die Elementtypen, die Beziehungstypen und die Constraints innerhalb eines Personenkreises oder einer Organisation gut vordefiniert sind. Die erzeugten Diagramme können jederzeit in verschiedenen Formaten wie PDF, BMP, EMF, GIF, HTML und JPG exportiert werden.

Um die Sichten einer Softwarearchitektur im yEd Graph Editor zu dokumentieren muss man zuerst überlegen, welche Elementtypen, Beziehungstypen sowie Constraints dafür notwendig sind. Man kann die Sichten als eine Sammlung von UML-Diagrammen dokumentieren, oder nicht standardisiert unter Verwendung anderer Elementtypen, Beziehungstypen und Constraints dokumentieren. Abbildung 27 zeigt ein mit dem yEd Graph Editor erzeugtes Anwendungsfalldiagramm, das ein Softwareszenario dokumentieren kann. Alle anderen Arten von UML-Diagrammen sind im yEd Graph Editor auch möglich.

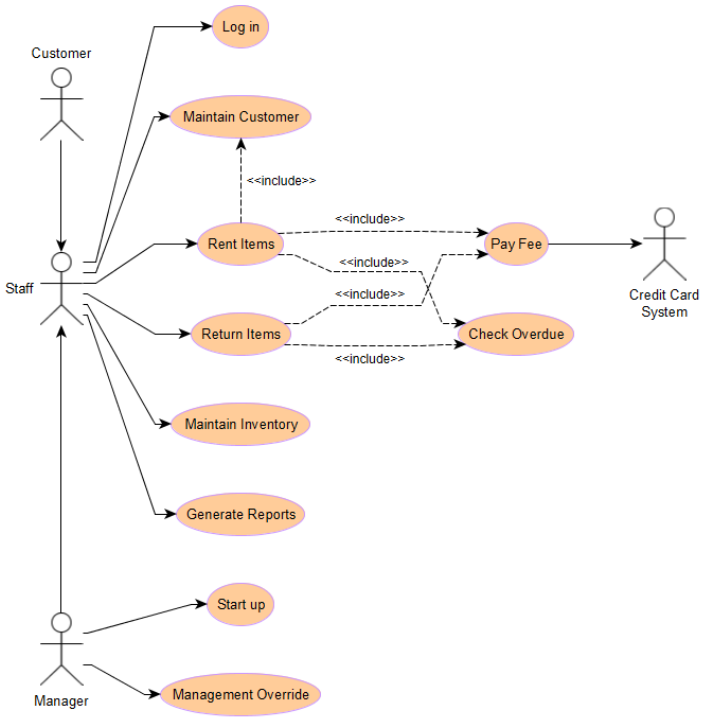

Abbildung 27: Das mit dem yEd Graph Editor erzeugte Anwendungsfalldiagramm

In der Abbildung 28 wird ein nicht standardisiertes Diagramm gezeigt. Dieses Diagramm beschreibt die Topologie eines Systems und kann als Dokumentation des

Physical Views verwendet werden.

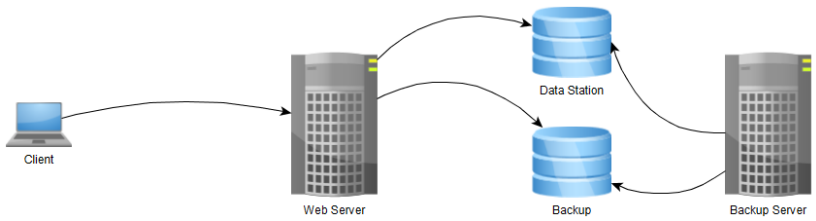

Abbildung 28: Mit dem yEd Graph Editor erzeugtes nicht standardisiertes Diagramm

4.3 VbMF

Im Oktober 2010 fand das Projekt INDENICA statt. Dieses Projekt zielte darauf ab, einen wichtigen Beitrag zum zukünftigen Internet durch die Bereitstellung von Virtual Service Plattformen (VSPs) zu leisten. Diese sollen die Realisierung der service-basierten Anwendungen unterstützen [17]. Als eines der Hauptziele des Arbeitspakets 3 der INDENICA wurde das View-based Modeling Framework (VbMF) vonder Forschungsgruppe Software Architecture (SWA) der Fakultät für Informatik der Universität Wien entwickelt [18]. Das VbMF kann verwendet werden, um die VSPs zu etablieren, zu bearbeiten sowie zu dokumentieren [42]. Obwohl das VbMF für die service-basierten Softwaresysteme entwickelt ist, ist es tatsächlich für die allgemeine Softwarearchitektur anwendbar.

Das VbMF basiert auf die IDE Eclipse und basiert auf modellgetriebener Architektur (MDA). Das heißt, es kann nicht nur die Softwarearchitektur dokumentieren sondern auch automatisch den Code generieren [31].

Die Diagramme von VbMF lassen sich in High-level Service Component View, Low-level Service Component View, High-level Service Deployment View und Low-level Service Deployment View unterteilen. (vgl. Abbildung 29) Der High-level Service Component View ist eigentlich ein C&C-Diagramm und benutzen Komponenten, Ports, Konnektoren und Annotationen um die wesentlichen Softwarekomponenten sowie die Beziehungen dazwischen darzustellen. Er wird durch den Low-level Service

Component View erweitert und verfeinert [31]. Der Low-level Service Component View ist für die service-basierten Softwaresysteme besonders gut anwendbar, da er zusätzliche Elementtypen, Ports und Konnektoren wie die Service Component Architecture (SCA) Komponente, den SCA Komponentendienst, den SCA Wire usw. einführt [18].

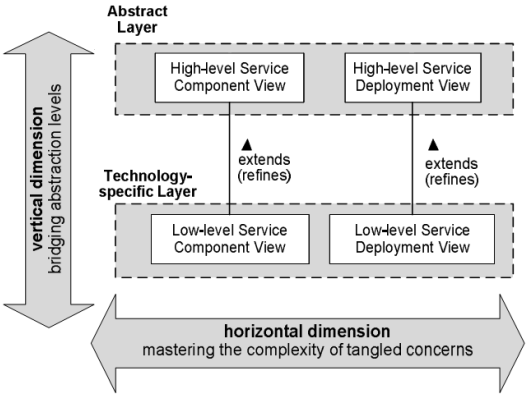

Abbildung 29: Die Diagrammarten im VbMF

(Quell: INDENICA, 2011, S. 10)

Der High-level Service Deployment View und der Low-level Service Deployment View sind dem UML Verteilungsdiagramm sehr ähnlich. Sie enthalten zwei Elementarten, Artefakte und Knoten. Ein Artefakt ist eine Softwarekomponente und ein Knoten ist ein Umgebungselement. Ein Artefakt und ein Knoten werden durch einen Konnektor „Deploy“ verbunden.

Im VbMF werden die Views nicht nur als Bäume sondern auch als Diagramme dokumentiert. Folgende Abbildung zeigt den Baum und das Diagramm eines High-level Service Component View.

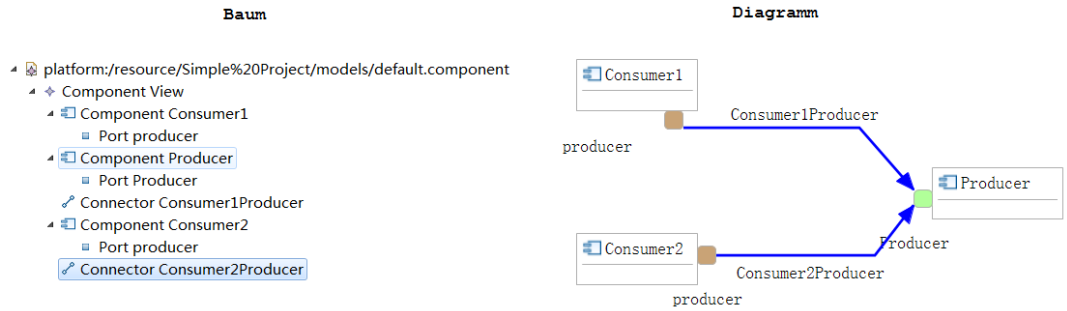

Abbildung 30: Die View-Formen im VbMF

Weil das VbMF für die service-basierten Softwaresysteme oder genauer gesagt für die VSPs entwickelt ist, fehlen viele Diagrammarten die für die Dokumentation von mancher Softwarearchitekturen sehr wichtig sein können. Beispielsweise können die Modulstile mit dem VbMF nicht dokumentiert werden. Deshalb ist die Fähigkeit von VbMF für die Softwarearchitektur auf die service-basierten Softwaresysteme beschränkt, aber der High-level Service Component View und die Service Deployment Views sind für die allgemeine Softwarearchitektur immer geeignet.

4.4 WebVbMF

Das WebVbMF ist eine webbasierte kollaborative Anwendung für die Softwarearchitekturdokumentation, es basiert auf den High-level Service Component View von VbMF, aber es ist nicht wie das VbMF speziell für die graphische Softwarearchitekturdokumentation von service-basierten Softwaresystemen entwickelt, es kann für die Bearbeitung von C&C-Diagrammen aller Softwaresysteme benutzt werden. Durch leichte Veränderungen und Ergänzungen können alle architektonischen Strukturen mit WebVbMF webbasiert und kollaborativ bearbeitet werden.

Um das WebVbMF zu benutzen braucht man eigentlich nur einen Browser, da es eine Webanwendung ist. Die Bedienungsanleitung von WebVbMF befindet sich im Anhang dieser Arbeit, die Schwerpunkte dieses Abschnitts sind die Beispiele von WebVbMF und die Unterschiede zwischen WebVbMF und den zuvor vorgestellten Tools.

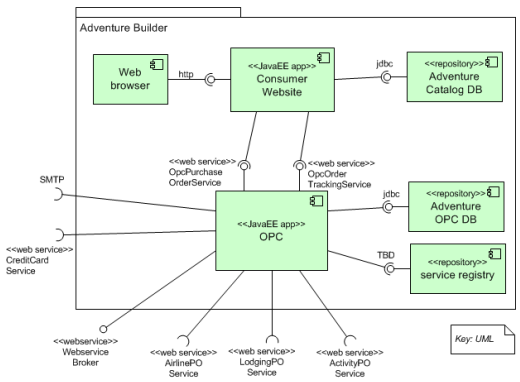

Abbildung 31: Das Komponentendiagramm des Adventure Builders

(Quell: https://wiki.sei.cmu.edu/sad/index.php/Top_Level_SOA_View)

Im Abschnitt 3.4.2 wurde die Struktur des Adventure Builders als ein Beispiel für den SOA-Architekturstil vorgestellt, wie das entsprechende UML-Komponentendiagramm des Adventure Builders in Abbildung 31 zeigt. Im Folgenden wird gezeigt, wie diese architektonische Struktur mit WebVbMF zu dokumentieren ist.

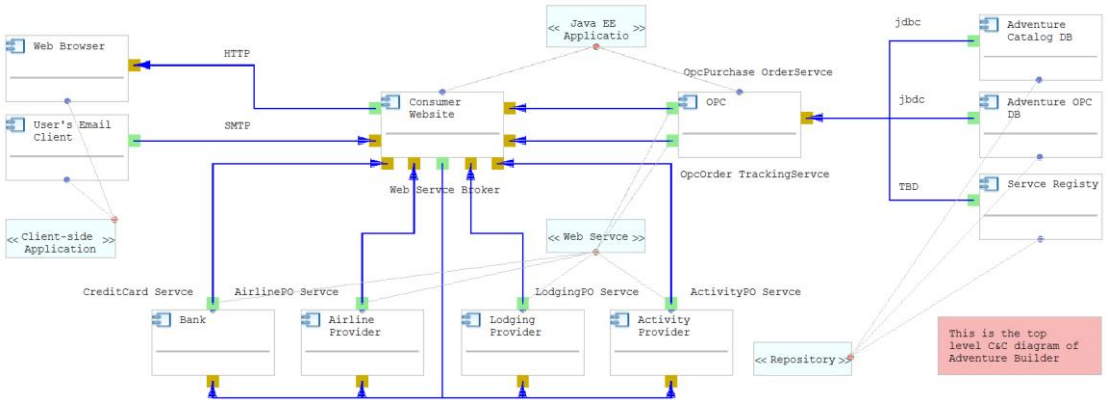

Abbildung 32: Das C&C-Diagramm von Adventure Builder

Abbildung 32 zeigt das mit WebVbMF erstellte C&C-Diagramm des Adventure Builders. Im Vergleich zum UML-Komponentendiagramm werden auch die clientseitigen Anwendungen und die Systeme der externen Dienstleistungsanbieter als Komponenten modelliert. Die Typen der Komponenten können mithilfe der ergänzenden Stereotyp-Elemente erklärt werden. Beispielsweise sind die Komponenten Consumer Website und OPC mit dem Stereotyp Java EE Application verbunden, dies besagt, dass diese beiden Komponenten Java EE Applikationen sind. Die Schnittstellen der Komponente im Komponentendiagramm können als Ports modelliert werden, z.B. die Schnittstelle TBD, durch die die Komponenten OPC und Servce Registy verbunden sind, kann im C&C-Diagramm von WebVbMF durch einen Eingangsport von OPC, einen Ausgangsport von Servce Registy und einen Konnektor, durch den die beiden Ports verbunden sind, modelliert werden. Die View Annotation im C&C-Diagramm dient dazu, das Diagramm zu erklären.

Wie im Abschnitt 3.1 schon angesprochen wurde ist Acme eine ADL. Mit Acme können unterschiedliche C&C-Diagramm erstellt werden. Folgende Abbildung zeigt ein mithilfe von Acme erstelltes C&C-Diagramm. Die Komponente MergeAndSort hat die

zwei Subkomponenten Merge und Sort. Diese Subkomponenten werden im WebVbMF aber nicht direkt unterstützt, deshalb wird in diesem Beispiel gezeigt, wie die Subkomponenten im WebVbMF zu dokumentieren sind.

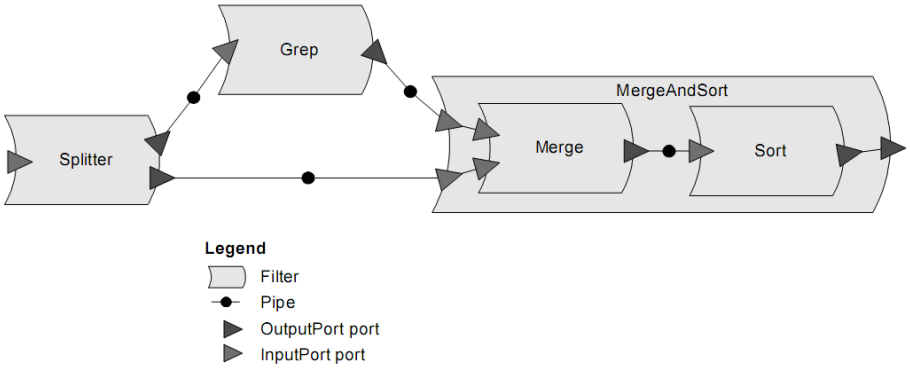

Abbildung 33: Mit AcmeStudio dokumentierte Struktur

(Quell: Ivers, J. et al., 2004, S. 39)

Das mit dem WebVbMF erstellte C&C-Diagramm wird in Abbildung 34 gezeigt. Die Filter werden als Komponenten und die Rohre werden als Konnektoren modelliert. Die Beziehung zwischen einer Komponente und seinen Subkomponenten kann durch eine Annotation dargestellt werden. Gleichzeitig müssen auch einige zusätzlichen Ports gebildet werden, durch die die Komponente mit seinen Subkomponenten verbunden werden kann. Die Beziehungen zwischen diesen Ports müssen ebenfalls durch eine Annotation beschrieben werden.

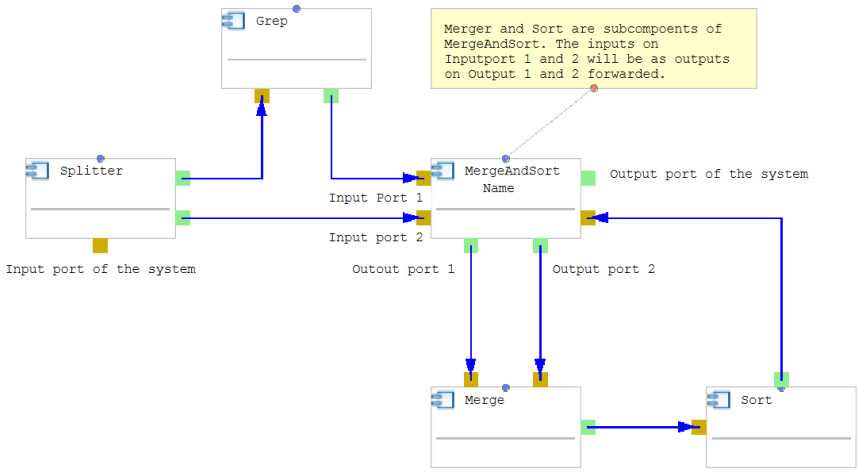

Abbildung 34: Das C&C-Diagramm des in der Abbildung 33 gezeigten Diagramms

Im Vergleich zu den zuvor vorgestellten Tools kann man mit dem WebVbMF kollaborativ die Softwarearchitekturen entwerfen und dokumentieren, das macht die Zusammenarbeit bei der Softwarearchitektur leichter. Obwohl in der RSA und VbMF man mithilfe von Concurrent Versions System (CVS), Git, usw. die Softwarearchitektur in Gruppen mitbenutzen lassen kann, bedeutet dies nicht, dass die RSA und das VbMF die Kollaboration unterstützen. Der yEd Graph Editor ist eine reine lokale Anwendung, die damit erzeugten Diagramme können weder mitbenutzt noch kollaborativ bearbeitet werden.

In Hinsicht auf der Fähigkeit zur Softwarearchitekturdokumentation ist die RSA offensichtlich besser als der yEd Graph Editor und das WebVbMF. Sie kann nicht nur standardisierte Diagramme sondern auch nicht standardisierte Topologien und Sketches erzeugen. Aber, durch einfache Veränderung und Ergänzung kann das WebVbMF auch andere Arten von Diagrammen unterstützen.

Der yEd Graph Editor ist flexibler als andere Tools, er erlaubt die externen Elementtypen, deshalb kann er theoretisch alle Arten von Diagrammen erzeugen.

In diesem Abschnitt werden vier Tools für die graphische Softwarearchitekturdokumentation vorgestellt. RSA vertritt die UML-basierten Anwendungen, und der yEd Graph Editor gehört zu Anwendungen die nicht standardisierte Diagramme erzeugen können. Das VbMF repräsentiert den Anwendungstyp, der speziell für die Dokumentation bestimmter Softwarearchitekturen geeignet ist. Diese drei Anwendungen können die Kategorien graphischer Architektur-Tools wiederspiegeln. Das WebVbMF basiert auf dem VbMF und betont die Zusammenarbeit bei der Softwarearchitektur. Es gibt viele andere graphische Architektur-Tools, aber jedes davon gehört zu einer der vier Kategorien und weist daher ähnliche Eigenschaften wie eines der drei vorgestellten Tools auf.

5 WebVbMF Implementierung

In diesem Kapitel wird die Implementierung des WebVbMFs beschrieben. Im Abschnitt 5.1 werden die Anforderungen an WebVbMF und die Herausforderungen für die Implementierung des WebVbMFs analysiert, dadurch kann das Problem der Implementierung des gesamten Systems in viele kleine Probleme, die einfacher gelöst werden können, unterteilt werden. Dann wird in jedem Abschnitt ein kleines Problem nach dem Questions, Options, and Criteria (QOC)-Ansatz gelöst. Diese Methode verfolgt genau dem „Teile und Herrsche“-Ansatz.

5.1 Anforderungen und Herausforderungen

Die Anforderungen an das WebVbMF sind:

- **Eine kollaborative Anwendung**. Das WebVbMF soll die Zusammenarbeit zwischen den Mitgliedern eines Teams garantieren. Dies bedeutet nicht nur die Möglichkeit, zusammen zu arbeiten, sondern auch die Module, die die Zusammenarbeit unterstützen, wie Projektmanagement und Teammanagement.

- **Eine webbasierte Anwendung**. Das VbMF ist eine lokale Anwendung. Die Hauptaufgabe des Projekts WebVbMF ist es, VbMF webbasiert zu machen. Diese Eigenschaft ist die Voraussetzung für die kollaborative Anwendung.

- **Persistenz**. Die Persistenz hier bedeutet, dass das WebVbMF dem Benutzer möglichweise eine gleiche Arbeitsumgebung liefern soll. Genauer gesagt, die im WebVbMF geöffneten Diagramme müssen beim Systemstart automatisch wieder geöffnet werden, sofern sie geöffnet können und nicht manuell geschlossen sind.

- **Zugriffskontrolle**. Die von WebVbMF erzeugten Diagramme können von unterschiedlichen Gruppen von Stakeholdern verwendet werden, (vgl. Tabelle 4 im Abschnitt 3.5) Um unerwartetes Verhalten zu vermeiden muss im WebVbMF eine Zugriffskontrolle implementiert werden.

- **Die Kompatibilität mit dem VbMF**. Die von WebVbMF erzeugeten Dokumente, in denen die Diagramme gespeichert sind, müssen auch aus VbMF bearbeitbar sein. Gemeinsam mit den Modulen von Download und Upload kann dies die Zusammenarbeit zwischen WebVbMF und VbMF sicherstellen.

- **Das Kontomanagement**. Im WebVbMF soll ein Subsystem implementiert werden, das sich mit der Verwaltung der Benutzer beschäftigt. Die Module wie Einloggen, Ausloggen, Registrierung und Modifizierung des Kontos müssen in diesem Subsystem implementiert werden.

- **Das Teammanagement**. Um die Zusammenarbeit zu unterstützen muss im WebVbMF ein Subsystem Teammanagement implementiert werden. Es soll die Etablierung neuen Team, die Verwaltung der Mitglieder und die Auflösung eines Teams ermöglichen.

- **Das Projektmanagement**. Die Diagramme im WebVbMF sollen als Projekte organisiert werden. Die normalen Dateioperationen wie das Umbenennen oder das Löschung sollen ermöglicht werden Zusätzlich dazu sollen entsprechende Module zur Unterstützung der Zusammenarbeit wie Download, Upload, Share oder Unshare auch implementiert werden.

Angesichts dieser Anforderungen, der vorhandenen Technologien und der Struktur des zu entwickelnden Projektes WebVbMF müssen folgende Herausforderungen bei der Implementierung bewältigt werden.

- **Die webbasierte Anwendung**. Der Schwerpunkt liegt darin, eine Technologie, die nicht nur für die Implementierung einer webbasierten Anwendung sondern auch für die Implementierung der Subsysteme von WebVbMF wie Kontomanagement, Teammanagement und Projektmanagement geeignet ist, zu finden.

- **Der Editor**. Bei der Implementierung des Editors muss man eine Technologie

finden, die sowohl für die Implementierung eines graphischen Editors geeignet ist, als auch mit der Technologie, mithilfe der das WebVbMF eine webbasierte Anwendung realisiert, kompatibel ist.

- **Die Dateioperationen**. Die Entscheidungen, wie die Dokumente der Diagramme zu organisieren, wie aufgrund der Dokumente ein Diagramm zu zeichnen ist und wie ein Diagramm in den Dokumente zu speichern ist müssen getroffen werden.

- **Die Kollaboration**. Ein für das gesamte System günstiger und mit der ausgewählten Web-Technologie kompatibler Mechanismus für die Zusammenarbeit ist hier der Schwerpunkt.

- **Die Persistenz**. Der Schwerpunkt liegt darin, wie der Status eines Diagramms (geöffnet oder geschlossen) zu entdecken ist sowie zu speichern ist und wie der Konflikte zwischen Persistenz und Kollaboration vermieden werden könnnen.

- **Die Zugriffskontrolle**. Die Fragen, welches Modell für das WebVbMF günstig ist und wie es implementiert werden soll, sollen beantwortet werden.

5.2 Webbasiertheit

Um das WebVbMF webbasiert zu machen müssen nicht nur das serverseitige Programm sondern auch das clientseitige Programm berücksichtigt werden. Mithilfe zusätzlicher Module, durch die das VbMF mit dem Server interagieren kann, kann das originale VbMF die Rolle von clientseitigen Programm des WebVbMFs spielen. Aber diese Realisierung hat folgende Nachteile:

- Wegen die Diversität der Plattformen, auf den das WebVbMF installiert und ausgeführt ist, muss das WebVbMF zumindest für jede mögliche Plattform speziell kompiliert werden, im schlimmsten Fall muss es mehrfach implementiert werden, ansonsten wird die Verwendungsmöglichkeit des WebVbMFs beschränkt.

- Der Benutzer muss das WebVbMF auf einem Gerät installieren.

Eine andere mögliche Form des clientseitigen Programms ist der Webbrowser. Eine Anwendung die beim Benutzer in einem Webbrowser abläuft bzw. dargestellt wird heißt Webanwendung. Wenn das WebVbMF als Webanwendung realisiert wird, können die oben genannten Nachteile beseitigt werden, und muss man sich um das clientseitige Programm gar nicht kümmern. Angesichts dieses Vorteilss ist das WebVbMF als eine Webanwendung realisiert.

Die möglichen Methoden zur Realisierung des serverseitigen Programmes sind z.B. Hypertext Markup Language (HTML), Servlet, RAP usw. HTML wurde von The World Wide Web Consortium (W3C) entwickelt. Die neuste Version ist HTML5. Sie wird verwendet, um die Webseiten, die von Webbrowsern dargestellt werden können, zu erzeugen. Ein HTML-Document enthält normalerweise einen Kopf und einen Körper. Im, Kopf können unterschiedliche Skripte, die komplizierte Logik realisieren können, enthalten sein. Der Körper definiert den in dem Webbrowser dargestellten Inhalt, der Text, Bilder, Hyperlink, Schalter, Skripten usw. enthalten kann [48]. Wegen der Möglichkeit, die Skripten und unterschiedlichen verwendbaren Elementtypen einzubetten, besitzt HTML die Fähigkeit, ein serverseitiges Programm als eine Menge von Webseiten zu implementieren.

Das Servlet wurde von Oracle für die Programmierung der serverseitigen Programme entwickelt. Es ist der eigentliche Bestandteil von Java-EE-Technologie. Im Wesentlichen ist ein Servlet eine spezielle Java-Klasse, die bestimmte Vorschriften einhalten muss. Das Servlet bietet die Möglichkeit, unterschiedliche HTTP-Anfragen zu behandeln und dem Client unterschiedliche HTTP-Antworten zurückzusenden. Mithilfe von Servlets und Java können komplizierte Webanwendungen entwickelt werden [30].

RAP ist ein Plugin-für Eclipse und ist für modulare Anwendungen, die von verschiedenen Arten von Clients wie Webbrowser, Rich Clients und mobile Geräte zugegriffen werden können, entwickelt [44]. Sie ermöglicht es, eine Webanwendung wie eine lokale Anwendung oder genauer gesagt wie Eclipse Plugin Projekte zu implementieren. Da einerseits die Plugin Development Enviroment (PDE) von Eclipse

viele nützliche Tools wie Standard Widget Toolkit (SWT), JFace usw. zur Verfügung stellt und andererseits mithilfe von RAP die Interaktionen zwischen den Servern und den Clients nicht berücksichtigt werden müssen, ist das RAP für die Entwicklung einer Webanwendung besonders geeignet [26]. Weil das RAP vorteilhafter als HTML und Servlet ist, wird das WebVbMF als ein Plugin-Projekt mithilfe von RAP und PDE implementiert, HTML und Servlets werden dagegen als ergänzende Technologien benutzt.

PDE ist die Essenz von Eclipse und ermöglicht Eclipse durch unterschiedliche Plugins zu erweitern sowie die existierenden Plugins und die Module für die Entwicklung einer Anwendung wiederzuverwenden. Die wichtigsten Begriffe in Zusammenhang mit PDE sind Erweiterung, Erweiterungspunkt und Equinox [53].

Eine Erweiterung ist eine Anwendung oder ein Moduls das bestimmte Funktionen mitbringen kann. Eine Erweiterung wird durch ihren Erweiterungspunkt von anderen Plugins identifiziert und aufgerufen. Möchte man beispielsweise einen eigenen Editor im Eclipse einbetten, kann man den Erweiterungspunkt org.eclipse.ui.editors benutzen um damit den Editor einzuklinken, der eigene Editor wird dann als Erweiterung mit dem Erweiterungspunkt verknüpft. In diesem Zusammenhang lässt sich die Analogie zu einer Steckdose (Erweiterungspunkt) und einem Stecker (Erweiterung) verwenden [53].

Die Erweiterungen und die Erweiterungspunkte sind in der Manifestdatei deklariert, aber das dynamische Laden der Erweiterungen ist durch die Plattform Equinox, die ab Version 3.0 von Eclipse eingeführt wurde, garantiert [53]. Equinox basiert auf der Open Service Gateway Initiative (OSGi)-Technologie und beschäftigt sich hauptsächlich damit, aufgrund der Manifestdatei die Erweiterungen während der Laufzeit dynamisch zu laden sowie zu entladen [45].

Eine Webanwendung, die die Technologie von RAP verwendet, muss normalerweise mindestens einen sogenannten Eingangspunkt (Entry Point), einen WorkbenchAdvisor und einen WorkbenchWindowAdvisor implementieren. Dies ist das Skelett einer solchen

Anwendung. Der Eingangspunkt ist der Eingang der RAP-basierten Anwendung, er funktioniert wie die Hauptklasse einer Java Applikation. Im WebVbMF ist nur ein Eingangspunkt implementiert, als Plugin Projekt muss dieser Eingangspunkt unter Verwendung des Erweiterungspunktes org.eclipse.rap.ui.entrypoint in der Manifestdatei registriert werden (vgl. Abbildung 35). Bei der Registrierung eines Eingangspunktes muss nicht nur die Klasse des Eingangspunktes sondern auch der gewünschte Universal Resource Locator (URL) der Webanwendung eingetragen werden.

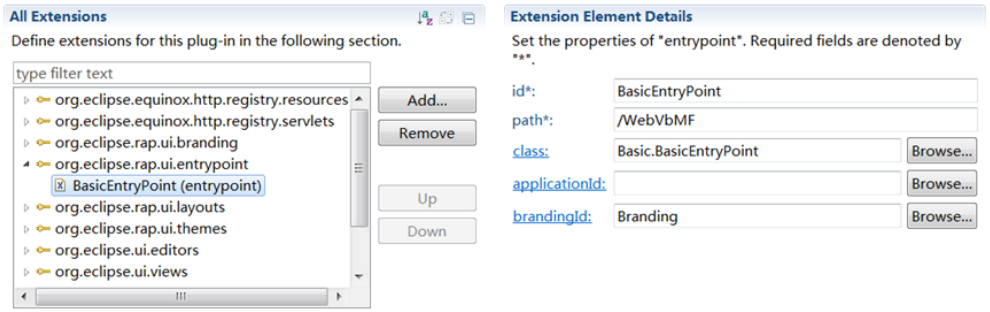

Abbildung 35: Die Registrierung des Eingangspunkts

Der WorkbenchAdvisor wird bei dem Start des Systems vom Eingangspunkt aufgerufen, und der WorkbenchWindowAdvisor wieder von dem WorkbenchAdvisorWebVbMFWorkbenchAdvisor. Der WorkbenchWindowAdvisor beschäftigen sich damit Workbench und WorkbenchWindow zu erstellen sowie zu konfigurieren. Dabei werden beispielsweise der Titel der Anwendung oder die Größe und Elemente der Benutzeroberfläche eingestellt. Sobald das Skelett implementiert ist, können alle anderen Subsysteme als unterschiedliche Elemente der Benutzeroberfläche auf diesem Skelett gebaut werden.

Als Plugin-Projekt kann das WebVbMF eine Eclipse-artige Benutzeroberfläche durch die Vererbung der Klassen des Pakets org.eclipse.ui einfach implementieren. Abbildung 36 zeigt die Elemente der Benutzeroberfläche von Eclipse.

Der größte Bereich der Benutzeroberfläche von Eclipse ist das Workbench-Window, das aus einer Menüzeile, einer Werkzeugleiste, einer unbestimmten Anzahl von Sichten, einer Statuszeile und dem Editor besteht.

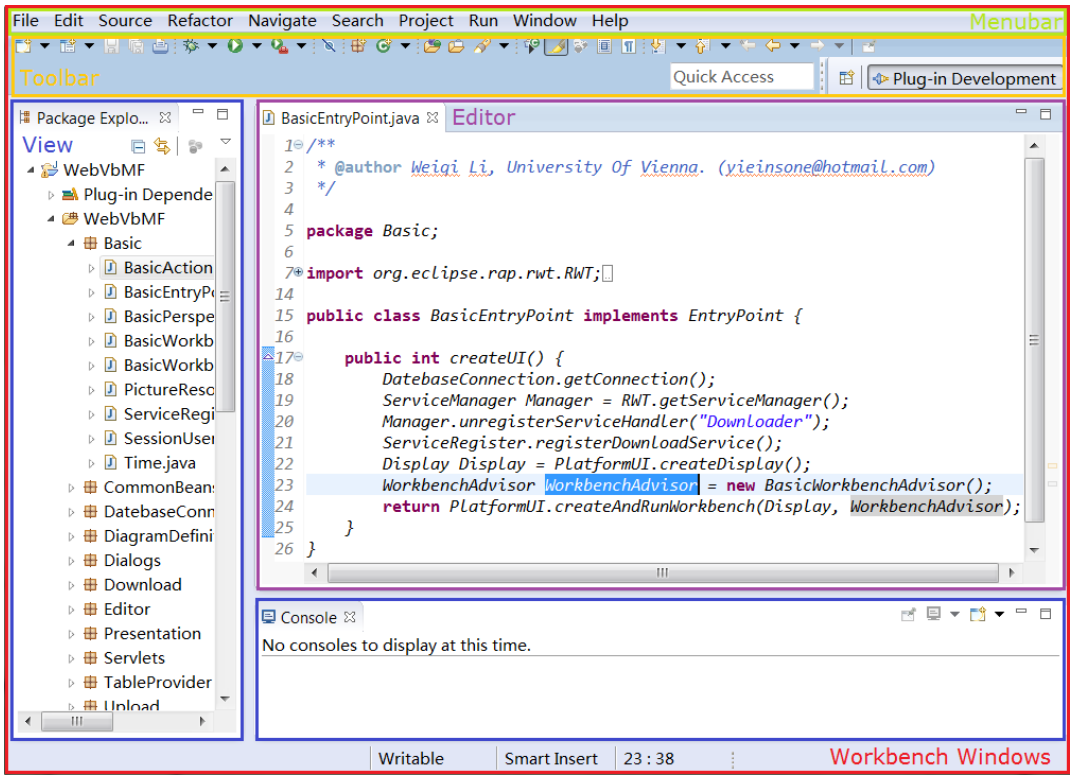

Abbildung 36: Die Bestandteile der Benutzeroberfläche von Eclipse

Eine bestimmte Kombination dieser UI-Komponenten heißt Perspektive. Eine Anwendung kann eine unbestimmte Anzahl von Perspektiven realisieren, aber zwei oder mehre Perspektiven können nicht gleichzeitig geöffnet werden. Beispielsweise sind im Eclipse die Perspektiven Java, Debug, Resource usw. vordefiniert. Ein anderer wichtiger Begriff ist der Assistent. Ein Assistent kann den Benutzer dabei unterstützen schrittweise eine vordefinierte Aktion durchzuführen. Die Assistenten sind z.B. der „New Project“-Assistent oder der „Export “-Assistent. Wichtig ist, dass solche Eclipse-artigen UI-Komponenten nur die erforderlichen Behälter definieren, die dahinter stehende Logik muss daher noch implementiert werden.

Fast jedes Subsystem von WebVbMF kann von den Benutzern in der Grafischen Benutzeroberfläche aufgerufen und gesteuert werden. Deshalb kann das WebVbMF als eine Sammlung von UI-Komponenten betrachtet werden. Für die Implementierung solcher Subsysteme werden die obigen UI-Komponenten verwendet. Folgende Abbildung zeigt ein Komponentendiagramm, das die Komponenten, die mit RAP und PDE im Zusammenhang stehen, enthalten.

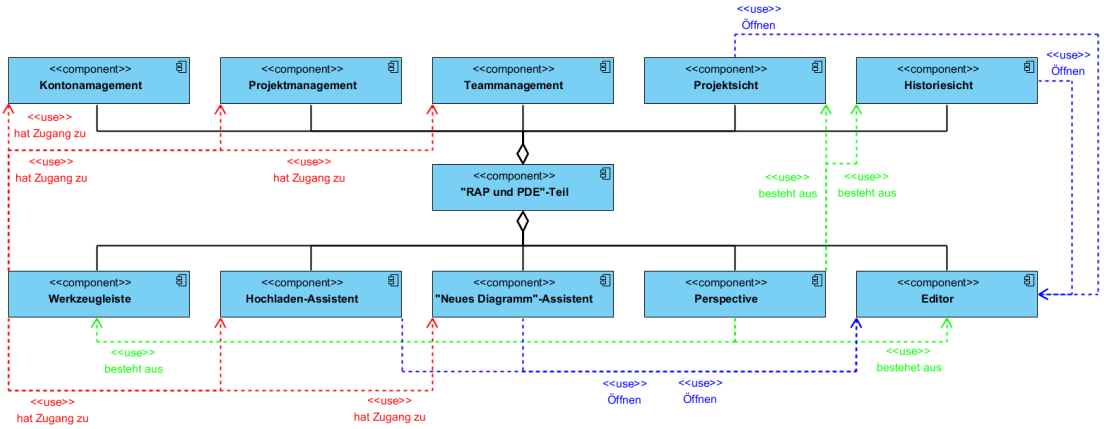

Abbildung 37: Das Komponentendiagramm einiger Subystemen

Die Abbildung zeigt, dass die mit RAP und PDE im Zusammenhang stehenden Kompomenten die Subsysteme Kontomanagement, Projektmanagement, Teammanagement, die Projektsicht, die Verlaufssicht, den Hochladen-Assistent, den „Neues Diagramm"-Assistent, die Werkzeugleiste, die Perspective und den Editor umfassen. Es gibt im WebVbMF nur eine Perspektive, die aus zwei Sichten, nämlich Projektsicht und Verlaufssicht, die aus Werkzeugleiste und dem Editor besteht. Die Werkzeugleiste enthält die Zugänge zum Upload-Assistent, dem „Neues Diagramm"-Assistent sowie den Subsystemen Kontomanagement, Projektmanagement und dem Teammanagement. Die Benutzer können den Editor in der Projektsicht, Verlaufssicht oder nach der erfolgreichen Ausführung des Upload-Assistenten oder des „Neues Diagramm"-Assistenten öffnen.

Nach dem Einloggen eines Benutzers werden alle mit ihm verbundenen Diagramme in der Projektsicht aufgelistet, durch Doppelklick auf den Name eines Diagramms kann er dieses Diagramm öffnen und dann bearbeiten. Die Verlaufssicht kümmert sich um die Veränderungen alle mitbenutzten Diagramme. Wenn eine Veränderung stattfindet, wird sie einen neuen Datensatz erzeugen und ihn dem Benutzer zeigen. Die Projektsicht und die Verlaufssicht sind als die zuvor erläuterte UI-Komponente Sicht implementiert. Da in Eclipse keine geeignete UI-Komponente für das Subsystem von Kontomanagement, Projektmanagement und Teammanagement gibt, werden sie als Dialogfenster realisiert. Wie schon gesagt wurde, definieren Eclipse UI-Komponenten nur die erforderlichen

Behälter, deshalb müssen alle in der Abbildung 37 aufgelisteten Komponenten mithilfe geeigneter Technologien bestimmten Inhalte besitzen. Zu diesen Technologien zählen Swing, Abstract Window Toolkit (AWT), SWT und JFace usw.

Swing und AWT sind Bestandteile von Java Foundation Classes (JFC). AWT ist die primitivste Graphical User Interface (GUI) in Java. Es ist thread-sicher, das heißt, man muss sich nicht um die Aktualisierung der Benutzeroberfläche kümmern. Zusätzlich dazu kann das AWT bestimmte Elemente der Benutzeroberfläche automatisch entfernen, wenn sie nicht mehr benutzt werden. Einige GUI-Komponenten, wie Tabellen oder Bäume werden von SWT nicht unterstützt. Außerdem kann das Aussehen und das Verhalten einer von SWT definierten GUI-Komponente sich aufgrund der Plattform verändern. Dies verursacht das Write Once, Test Everywhere (WOTE) Problem. Swing basiert auf dem AWT, deshalb unterstützt er auch die Garbage Collection. Die Nachteile vom AWT sind bei Swing ausgeglichen. Das heißt, dass einerseits Swing mehr GUI-Komponenten als SWT unterstützt und andererseits das Problem von WOTE in Swing gelöst ist. Aber die Probleme von Swing sind, dass es nicht thread-sicher ist und die Effizienz ein bisschen niedriger ist [15,28].

SWT und JFaces sind im PDE von Eclipse integriert, dies ist günstig für das WebVbMF, da es als ein Plugin Projekt entwickelt wird. Mithilfe von Java Native Interface (JNI)-Technologie kann das SWT indirekt die vom lokalen Betriebssystem definierten GUI-Komponenten manipulieren, deshalb können die Erscheinungsbilder und das Verhalten der GUI-Komponenten einer SWT-basierten Anwendung mit dem lokalen Betriebssystem übereinstimmen. Darüber hinaus ist SWT effizienter als der Swing und AWT [53]. JFaces basieren auf SWT und können als eine Erweiterung von SWT betrachtet werden. Sie kann das SWT nicht ersetzen, sondern zielt darauf ab, die Erzeugung einiger Arten von GUI-Komponenten, wie Tabellen und Bäume zu vereinfachen [15,28]. Angesichts dieser Vorteile werden SWT und JFaces für die Implementierung der mit RAP und PDE im Zusammenhang stehenden Komponenten verwendet.

Anderes als alle anderen in der Abbildung 37 aufgelisteten Komponenten ist der Editor schwer zu implementieren, da ein Grafikeditor, der mit dem RAP kompatibel ist, nur schwer zu realisieren ist, deshalb wird die Implementierung des Editors im folgenden Abschnitt isoliert vorgestellt.

5.3 Editor

Der Behälter des Editors kann einfach in der Perspektive definiert werden. In diesem Abschnitt wird die Implementierung des Inhalts des Editors vorgestellt. Da WebVbMF ein Plugin Projekt ist und die Technologie von PDE und RAP verwendet, muss einerseits der Editor durch einen entsprechenden Erweiterungspunkt, nämlich org.eclipse.ui.editors, eingeklinkt werden, andererseits muss man bei der Implementierung des Editors zuerst nachdenken, ob die zu verwendende Technologie für den Editor mit RAP kompatibel ist. Eine Technologie die mit RAP nicht kompatibel ist kann daher nicht benutzt werden. Beispielsweise ist das originale VbMF mithilfe von Graphical Modeling Framework (GMF) entwickelt worden, deshalb wäre es ideal für WebVbMF, bei der Implementierung des Editor auch GMF zu verwenden, weil dadurch das Code von VbMF wiederverwendet werden kann. Aber GMF ist mit RAP nicht kompatibel, GMF kann daher nicht genutzt werden. Im folgenden werden einige mögliche Technologien für den Grafikeditor analysiert.

Die erste Möglichkeit ist HTML5. In HTML5 sind drei Tags für das Malen im Webbrowser, nämlich Canvas, Vector Markup Language (VML) und Scalable Vector Graphics (SVG), definiert. Mit dem Tag Canvas kann den Behälter für die grafischen Elemente definiert, das Malen von den grafischen Elementen muss aber durch Javascript realisiert werden, das heißt, dass Canvas nur Application Programming Interfaces (APIs) für das Zeichnen liefert, aber nicht die Fähigkeit des Zeichnens hat. Da Canvas die pixelbasierte Farbwiedergabe verwendet, sind die Vergrößerung und die Verkleinerung von Elementen schwierig. Die Elemente der durch VML und SVG definierten Grafiken sind im Wesentlichen XML-Elemente. In beiden Technologien sind viele unterschiedliche Formen wie Linie, Oval, Rechteck usw. vordefiniert, darüber hinaus ist

die Benutzung von JavaScript in beiden Technologien möglich. Diese Eigenschaften ermöglichen es, komplizierte Bilder mithilfe von VML oder SVG zu zeichnen. Deshalb könnten beide Technologien für die Implementierung des Grafikeditors verwendet werden. Aber der Hauptnachteil dieser Technologien ist, dass nicht nur VML sondern auch SVG nur von bestimmten Webbrowsern gut unterstützt wird. Um die Verwendungsmöglichkeit des Grafikeditors zu erweitern müssen beide Technologien gleichzeitig benutzt werden, dies kann die Komplexität der Implementierung erhöhen [49].

Die andere Möglichkeit ist Graphical Editing Framework (GEF). Das GEF ist ein Plugin Projekt von Eclipse und wurde primär für den Grafikeditor sowie für baumartige Grafiken in Eclipse entwickelt. Das GEF verwendet die Model View Controller (MVC) Technologie, weshalb wenn man GEF benutzt. Es müssen nicht nur das notwendige Modell sondern auch die Wiedergabe von Grafiken definiert werden. Die Kontrollen sind die Brücken zwischen den Elementen des Modells und den Grafiken. Sie beschäftigen sich damit, alle Veränderungen im Modell zu entdecken sowie diese Veränderungen in den Grafiken wiederzugeben [14]. Die Kontrollen müssen somit auch definiert werden. Der MVC Mechanismus ist geeignet für die Trennung von Modell und Grafiken, aber er macht die Implementierung des Grafikeditors kompliziert. Wie GMF ist GEF mit RAP nicht kompatibel. Trotzdem kann die Technologie GEF in RAP benutzt werden, da ein Plugin[20] für RAP die Möglichkeit bietet, Code von GEF in RAP wiederzuverwenden. Das Problem dieses Plugins ist, dass es sich noch in der Entwicklung befindet und daher unerwartete Fehler auftreten können. Wenn dieses Plugin fertig entwickelt ist, gibt es noch das Problem der Zusammenarbeit von RAP und GEF. Da GEF auf dem Server ausgeführt ist, kann die Serverauslastung sehr hoch sein, wenn viele Clients den Editor benutzen.

JavaScript ist eine andere Möglichkeit für den Grafikeditor. JavaScript wurde im Jahr 1995 von Netscape veröffentlicht. Es wurde ursprünglich für dynamisches HTML

[20] http://eclipse.org/rap/incubator/

entwickelt, aber heute findet es auch außerhalb von Browsern Anwendung. Anderes als Personal Home Page (PHP) und Active Server Pages (ASP) wird JavaScript grundsätzlich auf den Clients ausgeführt, dies kann die optimale Serverauslastung sichern [11]. Als eine Skriptsprache mit objektorientierten Fähigkeiten kann das JavaScript nicht nur die Elemente des Grafikeditors des WebVbMFs sondern auch das Verhalten dieser Elemente definieren. Angesichts der Nachteile von Canvas, VML, SVG und GEF und der Vorteile von JavaScript ist der Grafikeditor des WebVbMFs in Javascript implementiert.

Als ein C&C-Editor muss der Editor von WebVbMF dem Benutzer die Möglichkeit bieten, *Komponenten*, *Ports*, *Konnektoren*, *Annotationen*, *Properties*, *Stereotypen* und *View Annotationen* zu erzeugen. Eine Komponente kann eine Anwendung, ein System, eine Dienstleistung oder eine Entität die bestimmte Funktionalität bietet sein. Eine Komponente kann eine unbestimmte Anzahl von Ports, die die Schnittstellen dieser Komponente repräsentieren, haben. Die Ports lassen sich in Eingangsports und Ausgangsports unterteilen. Ein Eingangsport und ein Ausgangsport können durch einen Konnektor, der den Weg der Interaktion beschreibt, verbunden werden. Die Annotationen, Properties und Stereotypen beschäftigen sich damit, die Komponenten, die Ports und die Konnektoren zu beschreiben oder deren Verhalten zu beschränken. Die View Annotationen werden für die Beschreibung des ganzen Diagramms benutzt. In dieser Arbeit werden die Annotationen, die Properties, die Stereotypen und die View Annotationen *ergänzende Elemente* genannt. Darüber hinaus ist auch eine *Palette* notwendig, die den Benutzern dabei helfen kann, unterschiedliche Elemente zu erzeugen. Sie muss auch einige Funktionen wie rückgängig, wiederholen, speichern usw. haben. Diese Arten von Elemente und die Palette sollen mithilfe von JavaScript implementiert werden. Viele unterschiedliche JavaScript Bibliotheken können die Implementierung erleichtern. Draw2D Touch[21] ist eine solche Bibliothek, die unterschiedliche komplexe Knoten und Kanten definiert. Aufgrund dieser Knoten und Kanten können die Elemente des Editors definiert werden. Da die Komponente der komplizierteste Knoten und der

[21] Ursprünglich Open Jacob Draw2D, http://www.draw2d.org

Konnektor die komplizierteste Kante ist, werden deren Implementierung im Folgenden vorgestellt.

Abbildung 38 zeigt ein Klassendiagramm, das die Klassen, die mit der Klasse *ComponentFigure* im Zusammenhang stehen, enthält. Die Klassen ComponentFigure, *SpecialInputPort*, *SpecialOutputPort*, *OutputPortFigure* und *InputPortFigure* sind die Unterklassen von der in Draw2D Touch definierten Klasse *Figure*. Eine Komponente kann viele Eingangsports und Ausgangsports haben, jeder Eingangsport/Ausgangsport besitzt genau eine Name-Box, in der der Name des Ports gezeigt wird. Jede Komponente hat noch einen speziellen Eingangsport, der die Komponente mit einer unbestimmten Anzahl von ergänzenden Elementen verbinden kann. Abbildung 39 zeigt die Zusammenhänge zwischen diesen Elementen.

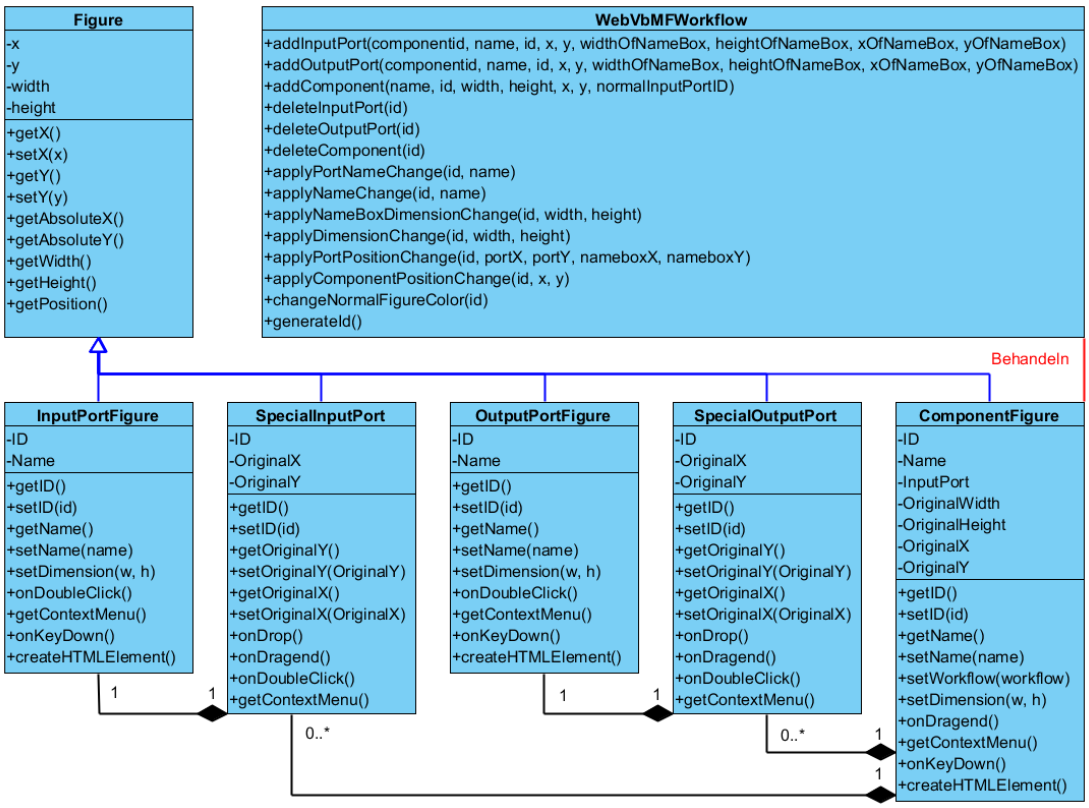

Abbildung 38: Das Klassendiagramm der Komponente

Wenn eine Komponente vergrößert, verkleinert oder verschoben wird, müssen die betroffenen Eingangsports, Ausgangsports, die Name-Boxs und der Eingangsport für die Annotationen repositioniert werden. Dies ist in der Methoden *setDimension(w, h)* und

onDragend() definiert.

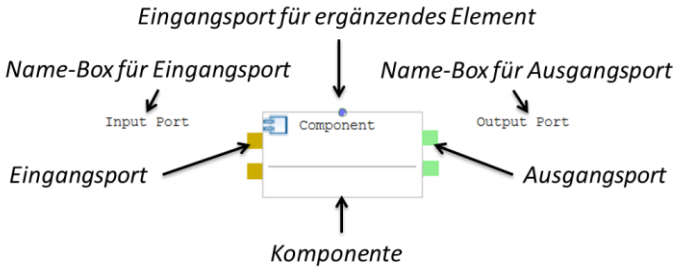

Abbildung 39: Die Komponente und die mit ihr verbundenen Elemente

Angesichts der Tatsache, dass ein Port sich immer an einem Rand einer Komponente befindet und deshalb nur eine der vier möglichen Formen von XY-Koordinaten, also (-8, y), (Component.width+8, y), (x, -8), (x, Component.height+8) aufweist, können bei der Größenveränderung einer Komponente dadurch die Ports repositioniert werden, dass sie wieder an demselben Rand der veränderten Komponente gelegt werden. Beispielsweise eine Komponente mit der Größe von 150*80 hat einen Port, der sich am rechten Rand der Komponente befindet und dessen genauen XY-Koordinaten (158, 50) sind. Wenn die Größe der Komponente auf 180*80 verändert wird, befindet sich der Port dann wieder am rechten Rand der Komponente und hat die neuen XY-Koordinaten (188, 50). Wenn die Position der Komponente verändert wird, werden die neuen Positionen von den Ports und den Name-Boxes unter Berücksichtigung des Offsets der Komponente und den originalen Positionen von Ports und Name-Boxes bestimmt. Beispielsweise wenn eine Komponente von (200,150) zu (300, 100) verschoben wird, ist der Offset (100, -50). Eine Name-Box von ihr, die sich in (150, 120) befindet, wird dann zu (150+100, 120-50), also (250, 70), bewegt.

Die Methoden *getContextMenu()* und *onKeyDown()* liefern den Benutzern die Möglichkeit, eine Komponente im Kontextmenü oder durch Drücken der Löschtaste zu löschen. Wenn eine Komponente gelöscht wird, sollen alle einschließenden Ports, Name-Boxs, Konnektoren und Verbindungen zwischen dieser Komponente und den ergänzenden Elementen gelöscht werden, falls es noch Verbindungen zwischen den zu löschenden Konnektoren und den ergänzenden Elementen gibt, sollen sie auch gelöscht werden.

In der Klasse *WebVbMFWorkflow* werden zahlreiche Methoden zur Behandlung der Komponente und dem Port, wie z.B. *addComponent(name, id, width, height, x, y, normalInputPortID)*, *deleteInputPort(id)*, *deleteComponente(id)* implementiert. Diese Methoden werden dann benutzt, um in der Palette neue Komponenten zu erzeugen oder während der Kollaboration die Komponente zu verändern.

Ein Konnektor ist im WebVbMF immer mit einer *Name-Box* und mehreren unsichtbaren *Dummy Figuren* verbunden. In der Name-Box wird der Name des Konnektors gezeigt. Da der Konnektor als eine Linie keinen Port haben darf sowie da es manchmal notwendig ist, durch die ergänzenden Elemente den Konnektor zu beschreiben oder dessen Verhalten zu beschränken, werden die Dummy Figuren zur Verbindung mit den ergänzenden Elementen eingeführt. Jede Dummy Figur hat genau einen Eingangsport, der mit den ergänzenden Elementen verbunden werden kann. Alle Dummy Figuren eines Konnektors besitzen immer die gleiche Identität. Das gleich gilt für deren Eingangsports, die Identität der Dummy Figuren und die Identität der Eingangsports der Dummy Figuren die mit der Identität von dem Konnektor im Zusammenhang stehen. Dadurch wird garantiert, dass wenn der Eingangsport einer Dummy Figur mit einem ergänzenden Element verbunden ist, man einfach entdecken kann, welcher Konnektor mit diesem ergänzenden Element verbunden ist. Die Beziehungen zwischen dem Konnektor und den einschließenden Elementen werden in Abbildung 40 gezeigt.

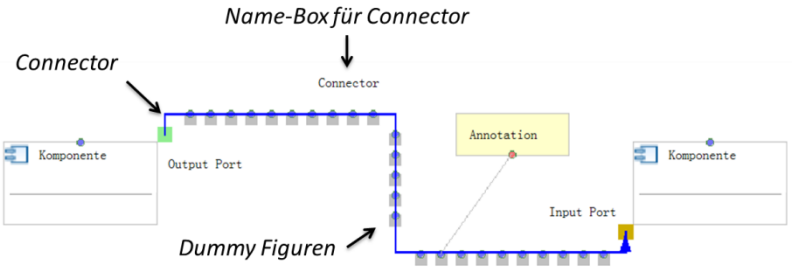

Abbildung 40: Der Konnektor und die Name-Box und Dummy Figuren von ihm

Das in der Abbildung 41 gezeigte Klassendiagramm verdeutlicht die Klassen, die mit dem Konnektor im Zusammenhang stehen. Die Klassen *SpecialInputPort*,

SpecialOutputPort, *NameBoxFigure* und *DummyConnectionFigure* sind Unterklassen der in Draw2D Touch definierten Klasse Figure. Die Klasse *SpecialConnection* ist Unterklasse der in Draw2D Touch definierten Klasse *Connection*.

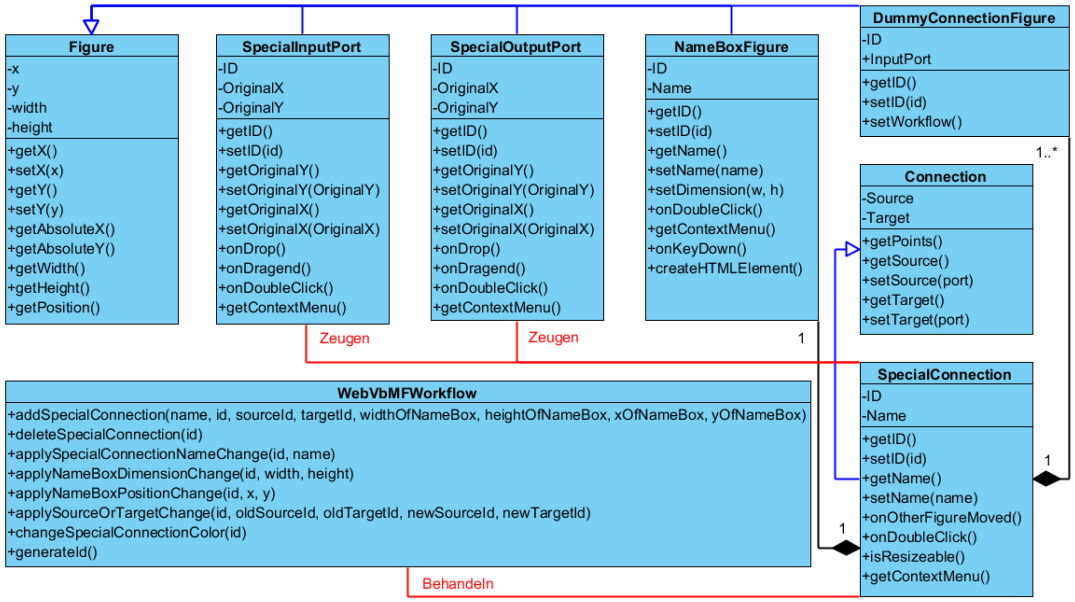

Abbildung 41: Das Klassendiagramm des Konnektors

Der Konnektor ist in der Klasse SpecialConnection definiert und die Dummy Figur und die Name-Box ist jeweils in DummyConnectionFigure und NameBoxFigure definiert. Wenn ein Benutzer einen Konnektor erzeugt, sollen die notwendige Name-Box und die notwendigen Dummy Figuren auch erstellt werden. Durch Aufruf der Methode *getPoints()*der Klasse Connection können alle Wendepunkte des Konnektors herausgefunden werden. Dann werden zwischen jeweils zwei Wendepunkte unter Einhaltung eines bestimmtem Abstands die Dummy Figuren erzeugt. Ein anderes Problem bei der Implementierung von Konnektor ist es, wie die Dummy Figuren gehandelt werden sollen, wenn die Länge, die Position oder die Gestalt des Konnektors verändert ist. Die Methode *onOtherFigureMoved()* beschäftigt sich damit, dieses Problem zu lösen. Die Methode dafür ist, wenn der Konnektor verändert ist, werden alle Dummy Figuren, die mit den ergänzenden Elementen nicht verbunden sind, gelöscht. Die verbundene Dummy Figur wird dann in der Mitte des Konnektors oder an einen Wendepunkt gelegt. Danach werden zwischen jeweils zwei Wendepunkte die Dummy Figuren neu erzeugt.

Ein Konnektor kann durch Drag&Drop des Ausgangsports, falls das Ziel ein Eingangsport ist, erzeugt werden, Diese Aktion kann auch umgekehrt ausgeführt werden. Dies ist in der Methode *onDrop()* der Klassen SpecialInputPort und SpecialOutputPort definiert.

In der Klasse WebVbMFWorkflow werden zahlreiche Methoden zur Behandlung des Konnektors wie *addSpecialConnection(name, id, sourceId, targetId, widthOfNameBox, heightOfNameBox, xOfNameBox, yOfNameBox)*, *deleteSpecialConnection(id)*, usw. implementiert. Diese Methoden werden grundsätzlich vom Server dann benutzt, um auf dem Client einen neuen Konnektor zu erzeugen sowie um den clientseitiges Konnektor zu verändern.

Das in der Abbildung 42 gezeigte Klassendiagramm sollen die Klassen, die mit der Palette im Zusammen stehen, verdeutlichen. Im WebVbMF sind die zwei unterschiedlichen Paletten *NormalPalette* und *PaletteForTry* realisiert. Sie sind Unterklassen von *Palette*, die wieder Unterklasse von der in Draw2D Touch definierten Klasse *ToolPalette* ist. Die Unterschiede zwischen diesen Paletten sind gering. Im Vergleich zu der NormalPalette fehlt der PaletteForTry die Methode *Save()*.

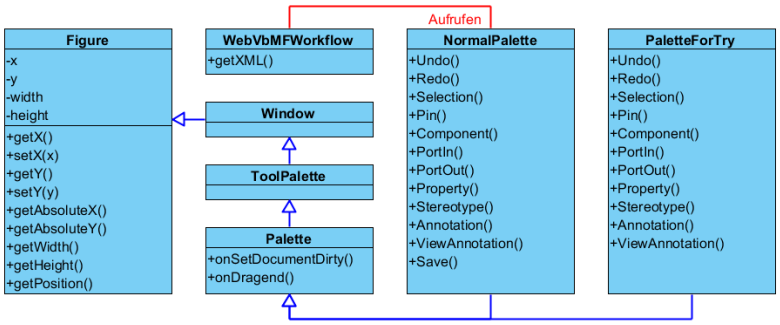

Abbildung 42: Das Klassendiagramm von Palette

Die wichtigste Funktion der Palette ist die Speicherung des Diagramms. Wenn ein Benutzer ein Diagramm erstellt, befindet das Diagramm auf dem Client, nur wenn er auf die „Save“-Schaltfläche in der Palette klickt, kann das Diagramm an den Server übertragen und auf den Server gespeichert werden. Bei der Speicherung des Diagramms wird von der Palette die Methode *getXML()* des WebVbMFWorkflow aufgerufen. Dadurch wird den Inhalt des aktuellen Diagramms als XML-Datenstrom temporär auf

den Client gespeichert. Um den gespeicherten XML-Datenstrom an den Server zu senden wird der *XMLHttpRequest* des Asynchronous JavaScript and XML (AJAX) verwendet. Der XMLHttpRequest ist der Grundbaustein der von AJAX und ermöglicht es, die Daten einer Webseite dynamisch vom Webserver abzurufen sowie an den Server zu senden, ohne dass dazu die Seite neu geladen werden müsste.

Eine weitere wichtige Funktion der Palette ist der sogenannte *Auswahlmodus*. Wenn das Diagramm sehr groß ist, wird es unübersichtlich. Der Auswahlmodus beschäftigt sich damit, dieses Problem zu lösen. Wenn der Auswahlmodus aktiviert ist und ein Element des Diagramms ausgewählt ist, werden die Transparenzwerte aller Elementen abzüglich des ausgewählten Elementes von 1,0 auf 0,5 gesenkt. Für alle Kanten, die mit dem ausgewählten Element nicht verbunden sind, wird ein Farbwechsel vollzogen. Dadurch soll besser überschaubar sein, mit welchen Elementen das gewählte Element verbunden ist.

Nachdem alle Elemente des Diagramms und die Paletten definiert sind, müssen die dadurch resultierten JavaScript-Skripte in den Editor eingebettet werden. Abbildung 43 zeigt das Klassendiagramm des Editors.

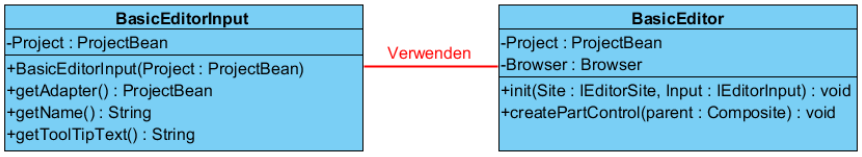

Abbildung 43: Das Klassendiagramm des Editors

Ein PDE- Editor besteht aus einem *EditorPart* und einem *EditorInput*. Der EditorInput wird von EditorPart bei der Initialisierung aufgerufen und kann den EditorPart darüber informieren, welches Diagramm im Editor zu öffnen ist. Die Methode *createPartControl(Composite parent)* der Klasse *BasicEditor* beschäftigt sich damit, den Inhalt des Editors zu definieren. Um die JavaScript-Skripte einzubetten wird innerhalb des Editors ein Browser implementiert. Anhand der von EditorInput gelieferten Informationen kann die Methode createPartControl(Composite parent) dynamisch die entsprechende URL generieren, diese URL wird dann im Browser geöffnet. Folgendes

Listing zeigt, wie solch eine URL generiert wird.

```
...
Browser = new Browser(parent, SWT.NONE);
if(ProjectName.equals("WebVbMFTry")&&DiagramName.equals("Enjoy it!")&&
                    Nickname.equals("WebVbMFTry")&&TeamName.equals("WebVbMFTry")){
     Browser.setUrl("/EditorTry");
}
else if(ProjectName.equals("WebVbMFHelp")&&DiagramName.equals("Get some help!")&&
                    Nickname.equals("WebVbMFHelp")&&TeamName.equals("WebVbMFHelp")){
     Browser.setUrl("/Welcome");
}
else{
     Browser.setUrl("/Editor?ProjectName="+ProjectName+"&DiagramName="+DiagramName
                    +"&Nickname="+Nickname+"&TeamName="+TeamName+"&NeedPalette="
                    +NeedPalette+"&ProjectView="+ProjectView);
}
...
```

Listing 1: Die Generierung einer Diagramm-URL für den Editor

Wie der Code gezeigt, der Editor ist auch für das Zeigen der Hilfeseite und der Seite für das Testen des WebVbMFs verantwortlich, deshalb dient der Editor lediglich dazu, einen Behälter für den Webbrowser zu liefern. Der Inhalt der durch die URL bestimmten Webseite wird durch ein Servlet dynamisch erzeugt, die Implementierung von diesem Servlet wird im folgenden Anschnitt vorgestellt.

5.4 Dateioperationen

Im letzten Abschnitt wurde die Implementierung des Editors vorgestellt. Wie anhand der Dokumente, in denen ein Diagramm gespeichert ist, ein Diagramm zu erzeugen und im Editor zu zeigen ist und wie ein Diagramm in die Dokumente zu speichern ist wirdein diesem Abschnitt vorgestellt. Abbildung 44 zeigt den Aufbau dieses Teilsystems.

Wenn ein Benutzer auf die „Save“-Schaltfläche in der Palette klickt, werden der XML-Datenstrom, der Projektname und der Diagrammname des Diagramms und alle anderen wichtigen Informationen als Parameter der URL an den Server übertragen. Diese Informationen werden von einem Servlet *DiagramSaver* behandelt und in ein Objekt von *DiagramBean* gespeichert. Die Informationen in dem Objekt werden dann mithilfe von *Writer* physisch in die entsprechenden Dateien gespeichert.

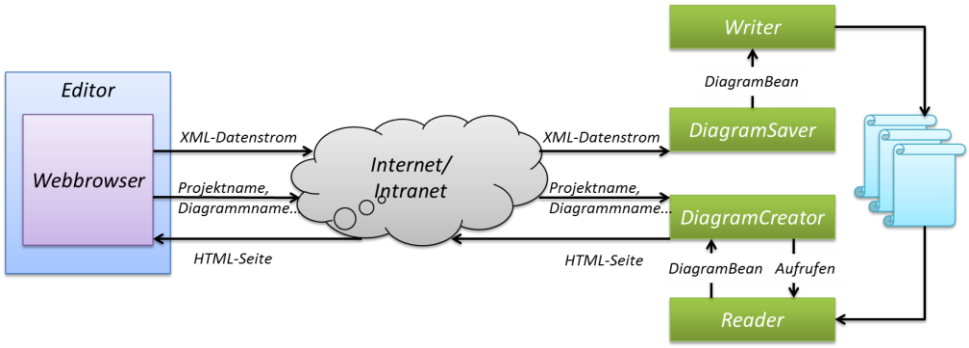

Abbildung 44: Der Aufbau des Teilsystems Dateioperationen

Wenn dagegen ein Benutzer in der Projektsicht oder in der Verlaufssicht durch Doppelklick ein Diagramm öffnet, werden der Projektname, der Diagrammname des Diagramms usw. als Parameter der URL an den Server übertragen. (vgl. Listing 1) Diese Informationen werden dann von einem Servlet *DiagramCreator* behandelt, durch Aufruf von *Reader* können die in der Dateien gespeicherten Informationen über das gewünschte Diagramm gesammelt und in ein DiagramBean Objekt gespeichert werdem. Anhand dieses Objekt wird der DiagramCreator eine Reihe von der in WebVbMFWorkflow (vgl. Abbildungen 38 und 41) definierten Weisungen erzeugen, diese Weisungen werden dann als Bestanteile von HTML-Seiten an den Client übertragen und werden dann auf dem Client durchgeführt, dadurch wird das Diagramm erstellt. Folgendes Listing zeigt den Inhalt der Diagram-Datei eines Diagramms als ein Beispiel.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Diagram>
    <Component>
        <X>273</X><Y>315</Y><Width>150</Width><Height>80</Height><Name>A</Name>
        <ID>eTzeNu5n4J</ID>
        <NormalInputPort><X>75</X><Y>0</Y><ID>M3imbCvzUT</ID></NormalInputPort>
        <SpecialOutputPort><X>158</X><Y>35</Y><Name>Out</Name><ID>33aLTz8hgt</ID>
        </SpecialOutputPort>
    </Component>
    <Component>
        <X>664</X><Y>289</Y><Width>150</Width><Height>80</Height><Name>B</Name>
        <ID>qupQZog844</ID>
        <NormalInputPort><X>75</X><Y>0</Y><ID>HM2IZ8o7EI</ID></NormalInputPort>
        <SpecialInputPort><X>-8</X><Y>29</Y><Name>In</Name><ID>6UyB8wNRI6</ID>
        </SpecialInputPort>
    </Component>
    <OutputPortNameBox>
        <X>451</X><Y>375</Y><Width>80</Width><Height>20</Height><Name>Out</Name>
        <ID>33aLTz8hgtOutputPortNameBox</ID>
    </OutputPortNameBox>
```

```
    <InputPortNameBox>
        <X>628</X><Y>253</Y><Width>80</Width><Height>20</Height><Name>In</Name>
        <ID>6UyB8wNRI6InputPortNameBox</ID>
    </InputPortNameBox>
    <NameBox>
        <X>508.5</X><Y>284.5</Y><Width>80</Width><Height>20</Height>
        <Name>Connector</Name><ID>yMvCZW1T5gSpecialConnection</ID>
    </NameBox>
    <SpecialConnection>
        <Name>Connector</Name><ID>yMvCZW1T5g</ID><Source>33aLTz8hgt</Source>
        <Target>6UyB8wNRI6</Target>
    </SpecialConnection>
</Diagram>
```

Listing 2: Die Diagram-Datei eines Diagramms

Wenn dieses Diagramm geöffnet wird, werden die Informationen in der Datei von einem Reader in ein DiagramBean Objekt transformiert, anhand des Objekt erzeugt der DiagramCreator folgende HTML-Seite.

```
<html>
  <head>
    <SCRIPT src='Draw2d.js'></SCRIPT>
    <SCRIPT src='Mootools.js'></SCRIPT>
    <SCRIPT src='Moocanvas.js'></SCRIPT>
    <SCRIPT src='Wz_jsgraphics.js'></SCRIPT>
    <SCRIPT src='Figures.js'></SCRIPT>
    <SCRIPT src='Palette.js'></SCRIPT>
    <SCRIPT src='Workflow.js'></SCRIPT>
  </head>
  <body style="margin:0px;padding:0px;">
    <div id="PaintArea" style="width:10000px;height:10000px" ></div>
    <SCRIPT type="text/javascript">
      var Workflow = new WebVbMFWorkflow("PaintArea");
      Workflow.setSnapToGeometry(true);
      Workflow.setEnableSmoothFigureHandling(true);
      Workflow.getCommandStack().addCommandStackEventListener(new CommandListener());
      var Listener = new SelectionMode(Workflow);
      Workflow.addSelectionListener(Listener);
      addWorkflowListener(Workflow);
      draw2d.Connection.setDefaultRouter(new draw2d.NullConnectionRouter());
      var Palette = new NormalPalette();
      Workflow.setToolWindow(Palette);
      Palette.setPosition(0,0);
      var ModeOfSynchoronisation = "Normal";
      var CanDirectDeletePort = 1;
      var ContextMenuController = 0;
      addComponent("A","eTzeNu5n4J",150,80,273,315,"M3imbCvzUT");
      Workflow.getCommandStack().execute(new addOutputPort("eTzeNu5n4J","Out",
        "33aLTz8hgt",158,35,80,20,451,375));
      addComponent("B","qupQZog844",150,80,664,289,"HM2IZ8o7EI");
```

```
    Workflow.getCommandStack().execute(new addInputPort("qupQZog844","In",
        "6UyB8wNRI6",-8,29,80,20,628,253));
    Workflow.getCommandStack().execute(new addSpecialConnection("Connector",
        "yMvCZW1T5g","33aLTz8hgt","6UyB8wNRI6",80,20,508.5,284.5));
  </SCRIPT>
 </body>
</html>
```

Listing 3: Die erzeugte HTML-Seite eines Diagramms

Diese HTML-Seite wird dann an den Client übertragen und auf dem Client durchgeführt, als Ergebnis wird im Editor des Clients folgendes Diagramm erzeugt.

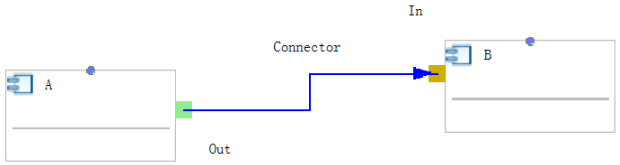

Abbildung 45: Das erzeugte Diagramm

Wie die Abbildung 44 zeigt ist das DiagramBean Objekt von die Brücke zwischen den Dateien und dem Diagramm, es beschäftigt sich damit, alle Informationen über ein Diagramm temporär im System zu speichern. Das Klassendiagramm von DiagramBean ist in der Abbildung 46 zu sehen. Eine Seite der Brücke, nämlich das im Editor gezeigte Diagramm ist schon erläutert, die andere Seite der Brücke, also die die Informationen über die das Diagramm aufrechterhaltenden Dateien, werden im Folgenden erklärt.

Das Diagramm ist im VbMF in einer Component-Datei und einer Componentx-Datei gespeichert. In der Component-Datei werden die Namen von sowie die Beziehungen zwischen Komponenten, Konnektoren und ergänzenden Elementen gespeichert, die Informationen für die graphische Darstellung des Diagramms wie Positionen, Größen usw. werden in Componentx-Datei gespeichert. Diese beiden Arten von Dateien sind eigentlich XML-Dateien und eng miteinander verbunden. Wenn ein Diagramm gespeichert wird, müssen immer beide Arten von Dateien behandelt werden. Wenn dagegen anhand der Dateien ein DiagramBean-Objekt erzeugt wird, müssen nach dem Auffinden der Informationen in beiden Arten von Dateien auch die Beziehungen zwischen den Informationen aus beiden Arten von Dateien analysiert werden. Diese

getrennte Speicherung macht es schwierig, die Dateien zu manipulieren, trotzdem werden sie auch im WebVbMF verwendet, um besser Kompatibilität mit dem VbMF sicherzustellen.

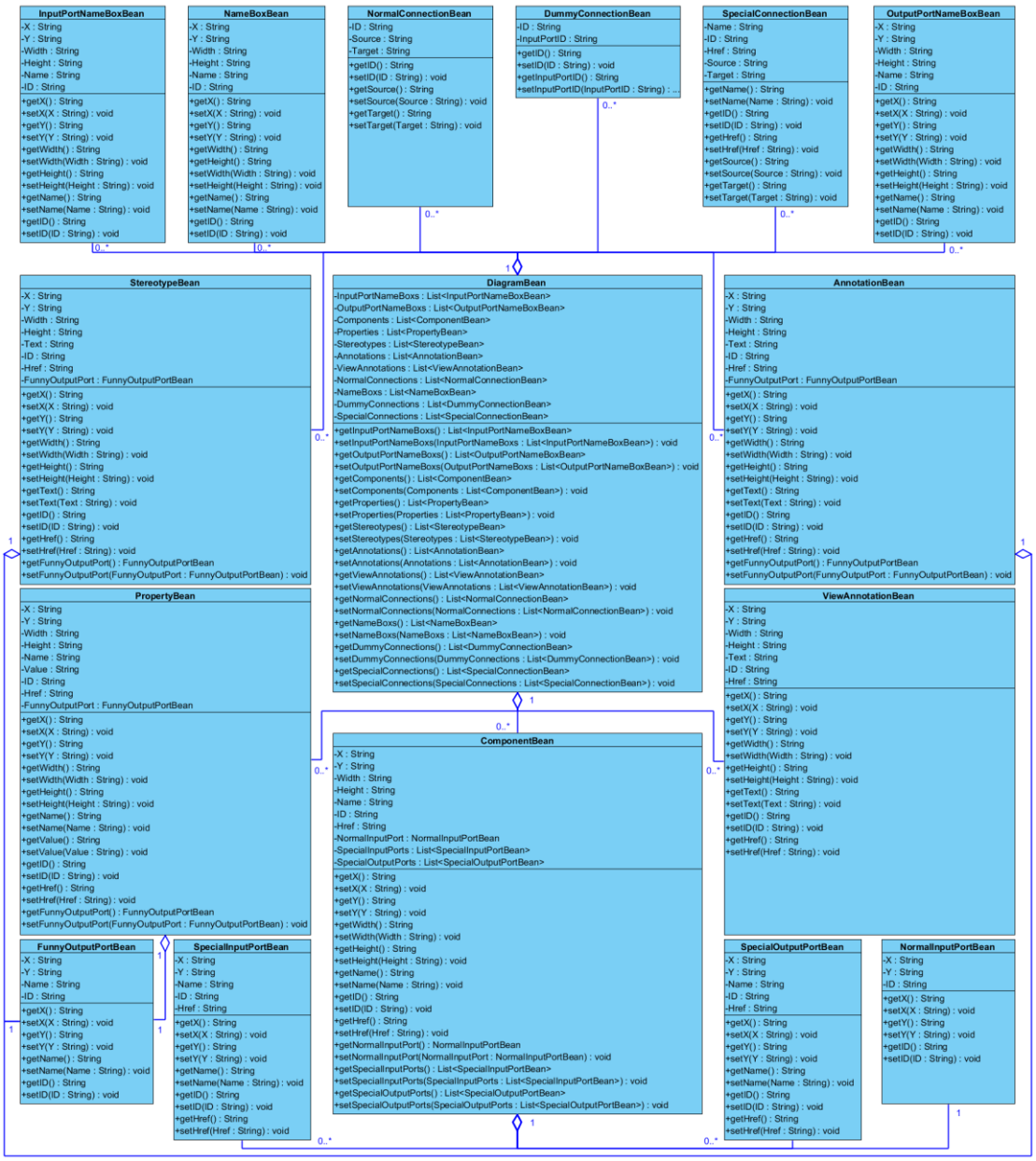

Abbildung 46: Das DiagrammBean Klassendiagramm

Zusätzlich dazu wird der neue Dateityp Diagramm-Datei eingeführt. Die Diagramm-Datei ist im Wesentlichen auch eine XML-Datei und ihre Struktur entspricht genau dem von dem Client gesendeten XML-Datenstrom. Ein Diagramm wird daher im WebVbMF nicht nur in Component-Datei und Componentx-Datei sondern auch

dupliziert in Diagramm-Datei gespeichert. Bei dem Auslesen von Daten wird zuerst überprüft, ob die Diagramm-Datei verfügbar ist. Wenn ja, wird die Diagramm-Datei benutzt, weil sie schneller und sicherer ist, sonst werden Component-Datei und Componentx-Datei verwendet.

Alle Dateien eines Diagramms sind XML-Dateien, sie können daher nicht direkt von den Servlets benutzt werden. Um die Informationen aus diesen Dateien in das DiagramBean Objekt zu transformieren ist ein XML-Parser notwendig. Heute gibt es viele unterschiedliche XML-Parser wie Document Object Model (DOM), Simple API For XML (SAX), Java-based Document Object Model (JDOM) und Document Object Model For Java (DOM4J).

DOM ist Plattform- und Sprach-neutral. Die Schnitttsellten von DOM liefern die Möglichkeit, die Informationen in einer XML-Datei zu manipulieren. Nach dem Parsing werden die Informationen baumartig im Speicher gespeichert. Der Hauptnachteil von DOM ist, dass für jedes Parsing immer die ganze Datei analysiert werden muss [27]. SAX ist ereignisgesteuert, das heißt, dass das Parsing kann gestartet oder gestoppt werden, wenn ein bestimmtes Ereignis eintritt, beispielsweise wenn ein Start-Tag gefunden wird. Deshalb braucht SAX für das Parsing nicht die ganze Datei, was SAX schneller macht als DOM. Die von SAX gefundenden Informationen werden nicht baumartig gespeichert [22]. JDOM funktioniert ähnlich wie DOM, aber es ist speziell für die Programmiersprache Java entwickelt und verwendet zahlreiche Java Technologien wie Java Collekcions [21]. Wie JDOM ist DOM4J auch speziell für Java entwickelt. DOM4J unterstützt viele Technologien über XML wie XML Path Language (Xpath), Extensible Stylesheet Language Transformations (XSLT). Darüber hinaus gibt es in DOM4J eine spezielle Schnittstelle für die Traversierung der XML-Datei, mit der alle Knoten einer XML-Datei sowie die Beziehungen zwischen den Knoten einfach gefunden werden können [8]. Da DOM4J einfacher, schneller und leistungsfähiger ist, wird für die Behandlung der XML-Dateien DOM4J benutzt.

Die in der Abbildung 44 gezeigten Writer und Reader sind solche, die ein

DiagramBean Objekt in XML-basierte Dateien oder die XML-basierten Dateien in ein DiagramBean Objekt transformieren. Writer ist der Oberbegriff für *ComponentWriter*, *ComponentxWriter* und *DiagramWriter*. Reader ist der der Oberbegriff für *ComponentReader*, *ComponentxReader*, *DiagramReader* und *ScriptXMLTraversal*. Darüber hinaus ist eine Klasse *DocumentGeter* als die Hilfsklasse von Reader implementiert. Das in der Abbildung 47 gezeigte Klassendiagramm soll die Beziehungen zwischen solchen Klassen veranschaulichen.

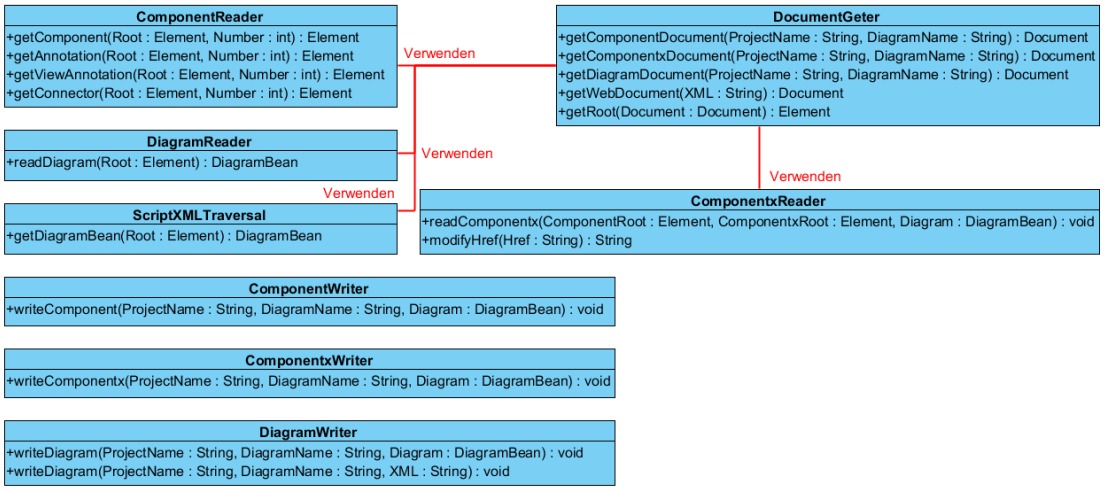

Abbildung 47: Das Klassendiagramm von den Readers und Writers

Wenn DOM4J eine XML-Datei einliest, wird die Datei baumartig in ein Objekt von *Document* gespeichert. Der DocumentGeter kann die Dateien eines Diagramms finden und dann die entsprechenden Document Objekte liefern. Die Wurzel eines baumartigen Document Objekt ist für die Analyse der Datei sehr wichtig, deshalb ist eine Methode *getRoot (Document : Document) : Element* auch implementiert. Die Methoden von DocumentGeter werden grundsätzlich von dem Reader aufgerufen.

5.5 Kollaboration

Eine kollaborative Anwendung soll die Zusammenarbeit in einer Gruppe unterstützen. Bei der Implementierung einer solchen Anwendung muss nicht nur ein Mechanismus für das Sharing der Dokumente, sondern auch ein Mechanismus für die Kollaboration entworfen werden.

5.5.1 Sharing-Mechanismus

Betrachten wir zuerst das Beispiel Gliffy[22]. Gliffy liefert dem Benutzer die Möglichkeit, im Webbrowser UML-Diagramme, BPMN-Diagramme, Ablaufdiagramme usw. zu generieren sowie die Diagramme zusammen zu nutzen. Wenn ein Benutzer ein Diagramm generiert, kann er durch den Eintrag der Email-Adresse eines anderen Benutzers dieses Diagramm mit diesem Benutzer zusammen genießen. Das heißt, die Diagramme können nicht automatisch für die Zusammenarbeit in einer Gruppe zur Verfügung gestellt werden.

Im WebVbMF ist ein anderer Mechanismus für das Sharing verwendet. Dieser Mechanismus basiert auf das Teammanagement und das Projektmanagement von WebVbMF. Es ist für jeden Benutzer von WebVbMF möglich, eine unbegrenzte Anzahl von Gruppen zu bilden und andere Benutzer in eigene Gruppen aufzunehmen. Wenn ein Benutzer Mitglied einer Gruppe ist, werden nicht nur seine eigenen Diagramme sondern auch die Diagramme dieser Gruppe für ihn zugänglich. Das Sharing der Dokumente ist nahtlos im WebVbMF integriert, oder mit anderen Worten, das WebVbMF ist für das Sharing entworfen. Wenn ein Benutzer mithilfe von Assistenten ein Diagramm generiert oder hochlädt, muss er entscheiden, ob dieses Diagramm privat sein muss oder in einer Gruppe mitbenutzt werden soll. Darüber hinaus können die Benutzer irgendwann im Projektmanagement ein privates Diagramm in einer Gruppe mitbenutzen lassen und ein schon mitbenutztes Diagramm wieder auf privat stellen oder in einer anderen Gruppe mitbenutzen lassen. Im Vergleich zu Gliffy betont das WebVbMF das Sharing in einer Gruppe, nicht das Sharing zwischen zwei Benutzern.

5.5.2 Kollaboration-Mechanismus

Der Kollaborations-Mechanismus ist der Schwerpunkt, er umfasst den Mechanismus für die Synchronisation und den Mechanismus für die Konfliktvermeidung. Darüber hinaus muss man bei der Implementierung eines kollaborativen Systems darüber

[22] http://www.gliffy.com

nachdecken, ob das System synchron oder asynchron kollaborativ sein muss. Es wird angenommen, dass ein Dokument von zwei Benutzer A und B eines kollaborativen Systems geöffnet ist. Wenn die Veränderung bei A gleichzeitig oder fast gleichzeitig automatisch auch bei B auftritt, ist eine synchrone Kollaboration zu beobachten, falls nicht, ist das System asynchron kollaborativ. Deshalb ist die Eigenschaft What You See Is What I See (WYSIWIS) die Schlüssel für die Klassifikation der Kollaboration [5].

Gliffy ist eine typische asynchrone kollaborative Anwendung. Die Veränderungen, die von einem Benutzer verursacht werden, werden nicht automatisch den anderen Benutzer gezeigt. Nur nach der Speicherung eines Benutzers kann anderer Benutzer durch manuelle Aktualisierung das neue Dokument sehen.

Das Dawn Projekt[23] ist ein Beipsiel für die synchrone kollaborative Anwendung. Dieses Projekt zielt darauf ab, GMF von Eclipse kollaborativ zu machen. Um dieses Ziel zu erreichen wird nicht nur ein Server sondern auch ein Plugin zur Erweiterung des lokalen GMFs implementiert. Zusätzlich dazu gibt es im Dawn Projekt auch eine Webanwendung *WebViewer*, die nicht nur die Rolle des zentralen Management spielt sondern auch die von den Benutzer generierten Diagramme zeigen kann. Wenn ein Benutzer im GMF-Editor ein Diagramm modifiziert, kann die Veränderung unmittelbar in anderem GMF-Editor oder in dem WebViewer gezeigt werden [12].

Obwohl die synchrone Kollaboration für die Zusammenarbeit in einer Gruppe ideal ist, darf man nicht sagen, dass sie unbedingt besser als die asynchrone Kollaboration, da sie zusätzliches Risiko bringt, das heißt, dass die Änderungen eines Benutzers durch die Änderungen anderen Benutzers ersetzt werden können. Der Nachteil von asynchroner Kollaboration ist, dass sie die gemeinsame Bearbeitung von Dokument nicht unterstützt. Die beste Lösung ist es, gleichzeitig beide Arten von Kollaborationen zu implementieren und dem Benutzer die Möglichkeit zu geben, irgendwann freiwillig eine für ihn geeignete Kollaboration zu wählen. Dies ist die grundlegende Methode für die Kollaboration im WebVbMF. Der Ersteller einer Gruppe hat das Recht, die Art der

[23] http://www.mftech.org/dawn/

Kollaboration dieser Gruppe zu ändern. Wenn die Mitglieder sicher und ungestört ein Diagramm bearbeiten und nach der Bearbeitung das Diagramm andere Mitglieder zeigen möchten, kann der Ersteller für diese Gruppe asynchrone Kollaboration wählen. Wenn das Diagramm gemeinsam bearbeitet werden soll, muss er dann synchrone Kollaboration aktivieren.

5.5.2.1 Asynchrone Kollaboration

Wenn die asynchrone Kollaboration aktiviert ist, ist es notwendig, den Zugang zu einem Diagramm zu sperren, während ein Benutzer dieses Diagramm bearbeitet. Dadurch kann die ungestörte Bearbeitung des Diagramms und die Konfliktvermeidung garantiert werden. Die Idee hier ist, wenn ein Benutzer ein Diagramm öffnet, wird es gesperrt und kann nicht mehr von anderen Benutzern geöffnet werden. Wenn er das Diagramm schließt, wird das Diagramm freigegeben und kann wieder von einem Benutzer geöffnet werden. Bei dem Öffnen eines Diagramms muss daher immer überprüft, ob das Diagramm gesperrt ist.

Um zu überprüfen, ob ein Diagramm schon geöffnet ist, wird ein *Überwacher* implementiert der immer im Hintergrund läuft und damit beschäftigt istdee Status (geöffnet oder geschlossen) alle Diagramme zu überwachen sowie sie in der Datenbank zu speichern, er hat auch die Aufgabe, die Symbole allen Diagramme bei der Statusänderungen zu aktualisieren. Diese Symbole sollen die Benutzer darauf hinweisen, ob die Diagramme momentan gesperrt sind oder nicht.

Dieser Überwacher ist eine serverseitige Komponente. Dank der Schnittstelle IPartListener2 der Eclipse API, die die Statusveränderungen aller UI-Komponenten von Eclipse wie Editor, Sichten, Menü, usw. entdecken kann, ist die Implementierung eines Überwacher nicht so kompliziert., Es gilt IPartListener2 zu implementieren und die Statusveränderungen der Editoren zu überwachen.

5.5.2.2 Synchrone Kollaboration

Die synchrone Kollaboration ist komplexer als asynchrone Kollaboration. Dazu muss ein Mechanismus für die Synchronisation und ein Mechanismus für die Konfliktlösung entwickelt werden. Die Synchronisation sorgt für die Konsistenz des clientseitigen Diagramms mit dem des Servers. Während der Synchronisation sollen die clientseitigen Veränderungen an den Server und die serverseitigen Veränderungen an den Client übertragen werden und sollen beide Diagramme entsprechend angepasst werden. Nach einer erfolgreichen Synchronisation sollen beide Diagramme identisch sein [12]. Ein Konflikt kann auftreten, wenn auf dem Server oder auf einem der teilnehmenden Clients eine Veränderung verursacht ist, da das System selbst nicht entscheiden kann, welche Veränderung vorgenommen werden soll. Die Konfliktlösung dient dazu, eine bestimmte Menge an Regeln, nachen den das System entscheiden kann, welche Veränderung vorgenommen werden soll, einzusetzen. Der Mechanismus für die Konfliktlösung steht normalerweise mit dem Mechanismus für die Synchronisation im Zusammenhang, deshalb dürfen die Synchronisation und die Konfliktlösung nicht isoliert betrachtet werden.

Für die Implementierung einer synchronen Anwendung kann die Technologie Operational Transformation (OT) benutzt werden. OT wurde erstmals im Jahr 1989 im System GRoup Outline Viewing Edit (GROVE) verwendet und wurde ursprünglich für die Konsistenzhaltung und Gleichzeitigkeitssteuerung beim kollaborativen Bearbeiten von Textdokumenten implementiert. OT gibt dem Benutzer die Möglichkeit, gleichzeitig Informationen hinzufügen, die bestehenden Informationen in einem Dokument zu modifizieren oder zu löschen [10].

Es wird angenommen, dass ein Objekt besteht aus der Zeichenkette ABC und es wird kollaborativ von den Benutzern 1 und 2 bearbeitet. Die möglichen Operationen O_n werden wie gefolgt definiert:

Insert(S,P): Einfügen S in die Position P,

Delete(S,P): Löschen S in der Position P,

Change(S,P): Ersetzen der Information in der Position P durch S.

Das Objekt soll nach der Durchführung von O_1 = Insert("X",0) und O_2 = Delete("C",2) den Inhalt von XAB haben. Ohne OT würde das Objekt aber den Wert XAC haben, die Ursache hierfür liegt darin, dass nach der Durchführung von O_1 die Position von C verändert wird. Deshalb beschäftigt sich die OT damit, die Operationen unter Berücksichtigung allen im Zusammenhang stehenden Operationen zu transformieren, sodass nach der Durchführung der transformierenden Operationen die erwarteten Veränderungen vorgenommen werden können. Die Transformation kann durch $T_n(O_1,O_2,,O_n)$ definiert werden. Wenn die Operationen O_1 und O_2 durchgeführt werden, gilt dann $O_1' = T_1(O_1,O_2)$ = Insert("X",0) und $O_2' = T_2(O_1,O_2)$ = Delete("C",3).[40] Graphisch wird dieses Beispiel in Abbildung 48 gezeigt.

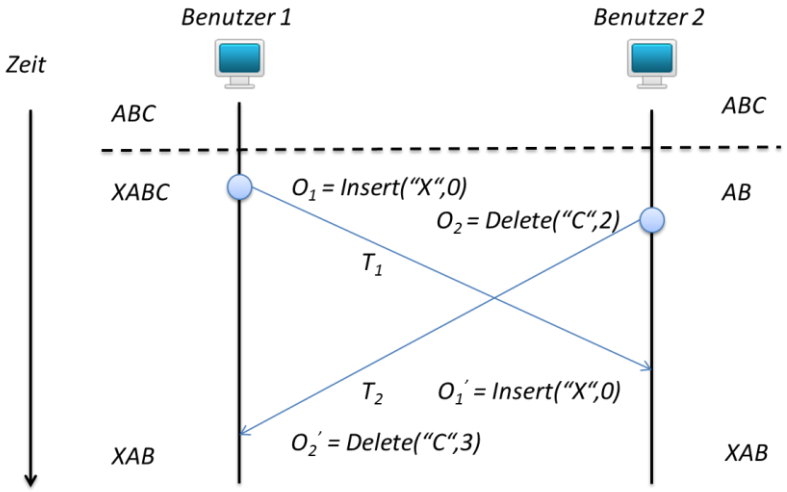

Abbildung 48: Ein Beispiel für OT

Es gibt viele unterschiedlichen Modelle für OT, wie CC[24], CCI[25] usw. Diese Modelle beschäftigen sich damit, den Mechanismus für die Transformation mittels OT zu entwickeln. Das CCI-Modell wird als ein allgemeiner Rahmen für die Konsistenzhaltung bei der Kollaboration betrachtet. Nach dem CCI-Modell wird dann die Konsistenzhaltung erreicht, wenn:

[24] Dieses Modell wurde im Jahr 1989 von Ellis, C. A. und Gibbs, S. J. veröffentlicht. CC ist die Abkürzung für Causality und Convergence [9].
[25] Causality Preservation, Convergence, Intention Preservation.

- die Kausalitätskonservierung, welche besagt, dass die Ausführungsreihenfolge der kausal abhängigen Operationen mit den natürlichen Kausalzusammenhängen zwischen diesen Operationen identisch sein muss. Der Kausalzusammenhang zwischen zwei Operationen wird einfach durch die „happened-before“-Beziehung zwischen diesen Operationen bestimmt. Beispielsweise werden drei Operationen in der Reihenfolge O_1, O_2, O_3 verursacht, dann muss bei der Ausführung diese Reihenfolge eingehalten werden,

- die Konvergenz, welche betont, dass alle transformierenden Operationen auf allen teilnehmenden Clients ausgeführt werden muss,

- sowie die Intentionskonservierung, welche sicherstellt, dass der Effekt der Ausführung einer Operation mit dem beabsichtigten Effekt dieser Operation identisch sein muss,

erfüllt sind [41].

Die Technologie OT ist weit verbreitet, sie wird beispielsweise in Google Docs[26] benutzt. Obwohl der Editor von WebVbMF für die Bearbeitung von Diagrammen, nicht Texten, implementiert ist, basieren die Diagramme aber auf XML. Das heißt, wenn ein Benutzer im WebVbMF ein Diagramm bearbeitet, bearbeitet er tatsächlich den XML-Text hinter diesem Diagramm. Aus diesem Grund kann OT auch für die Implementierung des Subsystems Kollaboration von WebVbMF verwendent werden. Unter Berücksichtigung der Tatsache, dass jedes Element des C&C-Diagramm eine einzigartige Identität besitzt und ohne OT richtig positioniert werden kann, wird für die Implementierung der Kollaboration für WebVbMF folgende einfache Methode benutzt.

Bei der synchronen Kollaboration können Konflikte dann auftreten, wenn Inkonsistenz zwischen den clientseitigen Diagrammen oder zwischen dem clientseitigen und dem serverseitigen Diagrammen auftritt. Beispielsweise besteht das serverseitige Diagramm aus den Elementen A, B und C. Falls der Inhalt des clientseitigen Diagramm

[26] http://www.google.at/intl/de-de/about/products/

A und B ist, kann die Ursache der Inkonsistenz das Einfügen des Elementes C in das serverseitige Diagramm oder das Löschen des Elementes C aus dem clientseitigen Diagramm sein. (vgl. Abbildung 49) In diesem Fall kann das System nicht selbst entscheiden, ob das Element C gelöscht werden soll. Diese Art von Konflikt kann mittels unterschiedlicher Synchronisationsformen der Synchronisation gelöscht werden.

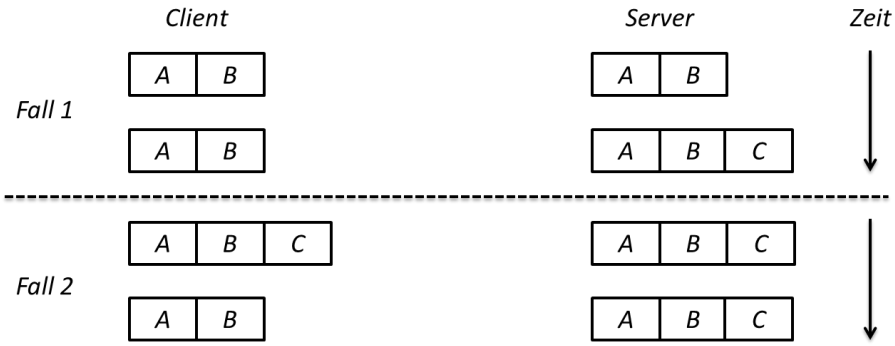

Abbildung 49: Mögliche Ursachen der Inkonsistenz

Abbildung 50 zeigt die Synchronisation bei WebVbMF. Die Synchronisation lässt sich in *Veröffentlichen* und *Aktualisieren* unterteilen. Sowohl beim Veröffentlichen als auch bei dem Aktualisieren werden die gesamten Daten des clientseitigen Diagramms an den Server übertragen und als Antwort sendet der Server dem Client gar keine Daten des Diagramms, sondern einige ausführbare Weisungen, die später auf den Clients ausgeführt werden, damit die beiden Diagramme wieder in einen konsistenten Zustand gebracht werden.

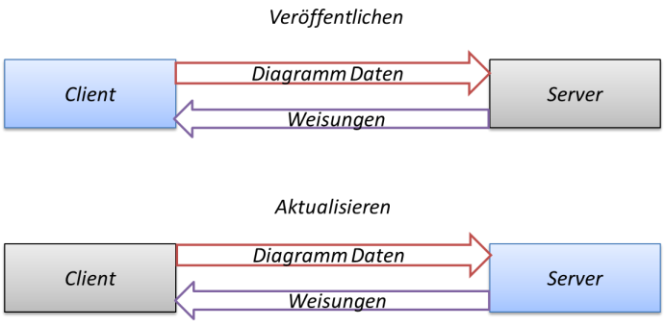

Abbildung 50: Veröffentlichen und das Aktualisieren

Bei der Synchronisation gilt es zwischen den zwei Rollen *Master* und *Slave* zu unterscheiden. Wenn das clientseitige Diagramm verändert wurde und der Client den

Server über die Veränderungen informiert, übernimmt der Client die Rolle des Masters und wird am serverseitigen Diagramm die clientseitigen Veränderungen vornehmen. Diese Art der Synchronisation wird Veröffentlichen genannt. Wenn das clientseitige Diagramm unverändert bleibt, übernimmt der Client die Rolle des Slave und das clientseitige Diagramm muss die serverseitigen Veränderungen vornehmen. Diese Art der Synchronisation heißt Aktualisieren. Die Unterschiede zwischen diesen beiden Rollen sind für jede Synchronisation wichtig, da sie bestimmen, wer das Recht hat, das Diagramm zu verändern. Durch diesen Mechanismus kann die Ursache der Inkonsistenz bestimmt werden, und ein konsistenter Zustand kann wieder herbeigführt werden. Folgende Abbildung soll die Unterschiede zwischen Master und Slave verdeutlichen.

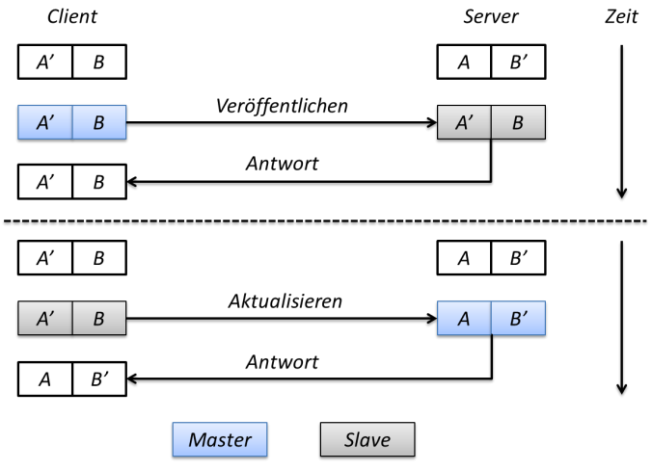

Abbildung 51: Unterschiede zwischen Master und Slave

Am Beginn sind beide Diagramme unterschiedlich. Das Diagramm des Clients besteht aus A‘ und B, dagegen sind A und B‘ die Elemente des serverseitigem Diagramms. Wenn der Client seine Veränderungen veröffentlicht, ist er Master, nach der Synchronisation haben das clientseitige und das serverseitige Diagramm den gleichen Inhalt, also A‘ und B. Wenn der Client seines Diagramm aktualisiert, ist er Slave, und der Inhalt seines Diagramm wird nach der Synchronisation durch A und B‘ ersetzt.

Bei der Kollaboration mittels WebVbMF werden die Benutzer nicht direkt miteinander kommunizieren um Konsistenz zu erreichen. Sie kommunizieren indirekt über den Server. Folgende Abbildung beschreibt die Kommunikationen zwischen den Benutzern bei der Kollaboration. Wenn Client 1 eine Veränderung vornimmt, wird ein

Veröffentlichung durchgeführt, danach entspricht das serverseitige Diagramm dem von Client 1. Die Diagramme von Clients 2, 3 und 4 stimmen erst dann mit dem Diagramm von Client 1 überein, wenn sie aktualisiert werden.

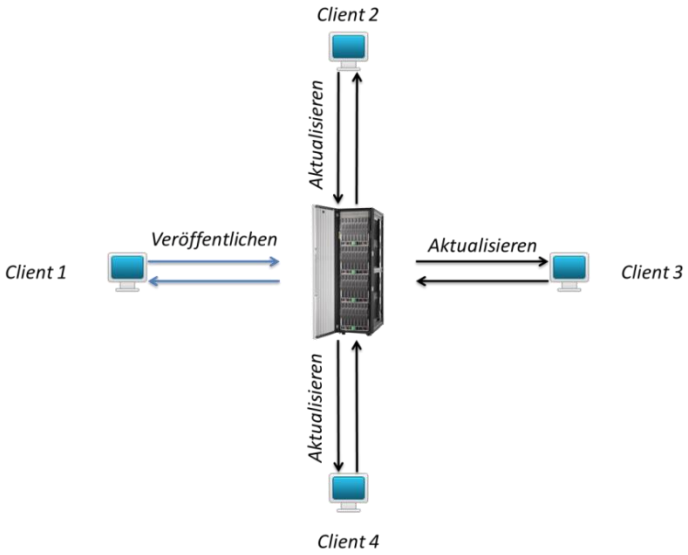

Abbildung 52: Kommunikationen zwischen den Benutzer bei der Kollaboration

Wenn zwei oder mehre Benutzer „gleichzeitig“ das selbe Element des Diagramm bearbeiten, werden die Veränderunger allen Benutzer anhand der Zeitpunkte der Synchronisationen sequenziell gespeichert, als Ergebnis sind nur die Veränderungen des Benutzers, der am spätesten synchronisiert hat, von Bedeutung. Ein Beispiel für das gleichzeitige Veröffentlichen wird in Abbildung 53 gezeigt.

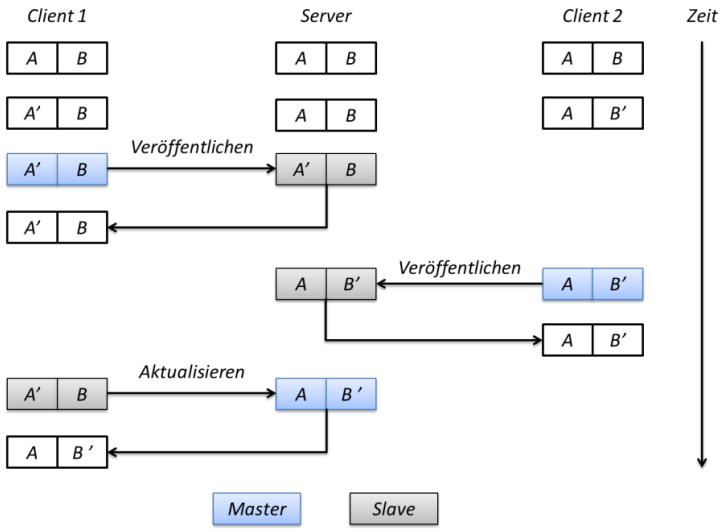

Abbildung 53: „Gleichzeitige“ Veröffentlichung

Am Anfang verfügen der Server und die Clients über das gleiche Diagramm. Nach der Bearbeitung hat das Diagramm von Client 1 den Inhalt von A‘ und B, dagegen enthält das Diagramm des Client 2 A und B‘. Wenn Client 1 und 2 „gleichzeitig“ die Veränderungen veröffentlichen, haben sie die Rolle des Masters und können den Inhalt des serverseitigen Diagramms ändern. Trotzdem wird kein Konflikt auftreten, weil es unmöglich ist, dass Clients 1 und 2 tatsächlich gleichzeitig die Veränderungen veröffentlichen. Wenn Client 1 früher veröffentlicht, werden seine Veränderungen am Server vorgenommen und das Diagramm von dem Server besteht aus A‘ und B. Dann werden die Veränderungen von Client 2 am dem Server vorgenommen und das Diagramm von dem Server besteht aus A und B‘. Wenn das Diagramm von Client 1 aktualisiert wird, wird es auch den Inhalt von A und B‘ besitzen. Das heißt, die Veränderungen von Client 1 werden durch die Veränderungen von Client 2 ersetzt.

Die nächste Frage ist, wie die synchrone Kollaboration durch WebVbMF unterstützt wird, oder mit anderen Worten, welche Softwarekomponenten für die Kollaboration verantwortlich sind und wie sie zusammen arbeiten. Folgende Abbildung skizziert die Bestandteile der synchronen Kollaboration und deren Beziehungen.

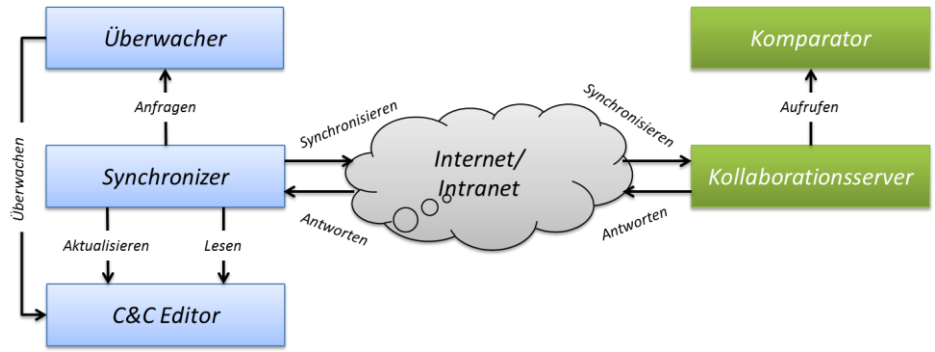

Abbildung 54: Aufbau des Subsystems Kollaboration

Wie die Abbildung zeigt besteht die synchrone Kollaboration von WebVbMF aus einem *Überwacher*, einem *Synchronizer*, einem *Kollaborationsserver* und einem *Komparator*. Der Überwacher ist eine clientseitige Komponente und läuft stetig im Hintergrund. Er hat die Aufgabe, den Inhalt des clientseitigen Diagramms zu überwachen. Wenn der Inhalt verändert ist, wird das Diagramm von ihm als „verändert“ von. Der Überwacher sorgt daher nicht für die Art und Weise einer

Veränderung, sondern dafür, ob eine Veränderung aufgetreten ist.

Der Synchronizer ist auch eine clientseitige Komponente, er übernimmt die Aufgabe, alle 3 Sekunden den Kollaborationsserver zu kontaktieren, um das clientseitige Diagramm mit dem Server zu synchronisieren. Wie schon erwähnt hat sind die Rolle der Benutzer und die Form der Synchronisation für eine Synchronisation sehr wichtig, deshalb muss der Synchronizer dem Kollaborationsserver solche Parameter mitteilen, damit der Server den richtigen Prozess durchführen kann. Die Rolle der Benutzer kann der Synchronizer durch Aufruf des Überwachers bestimmen. Wenn das Diagramm verändert ist, soll ein Veröffentlichen durchgeführt, sonst soll ein Aktualisieren stattfinden. Was noch fehlt sind die Daten des Diagramms. Sie können durch den Aufruf der Methode getXML() der Klasse WebVbMFWorkflow gesammelt werden (vgl. Abschnitt 5.3). Die Daten des Diagramms und die Parameter werden dann mithilfe von AJAX via XMLHTTP zum Kollaborationsserver übertragen. Der Synchronizer ist auch für die Aktualisierung des lokalen Diagramms verantwortlich. Die von dem Kollaborationsserver zurückgesandten Weisungen werden von ihm durchgeführt und das lokale Diagramm wird dadurch aktualisiert.

Der Kollaborationsserver ist ein serverseitiges Servlet. Er beschäftigt sich damit, von dem Synchronizer gesendete Diagramm mithilfe des Komparators mit dem des Servers zu vergleichen sowie anhand der Ergebnisse des Vergleichs die entsprechenden Weisungen, die das Diagramm von Clients aktualisieren, zu generieren und sie dann als HTTP-Antwort dem Client zu senden.

Der Komparator ist eine serverseitige Komponente. Die einzige Aufgabe des Komparators ist es, die Unterschiede zwischen zwei Diagrammen herauszufinden.

5.5.3 Verlaufssicht

Wenn ein Diagramm in einer Gruppe nutzt wird, ist es sinnvoll, den Mitgliedern die Möglichkeit zu geben, die Änderungsverlauf einsehen zu können. Die Änderungsabfolgen aller genutzten Diagramme werden in der Verlaufssicht aufgelistet.

Wenn ein Mitglied ein Diagramm nutzt, privatisiert, umbenennt, löscht, oder modifiziert, wird ein neuer Eintrag generiert. Falls das entsprechende Diagramm verfügbar ist, kann es durch Doppelklick geöffnet werden. Die Verlaufssicht kann nicht nur für die synchrone Kollaboration sondern auch für die asynchrone Kollaboration unterstützt werden.

Falls ein Benutzer einen alten Diagrammstand öffnet, kann er es nicht bearbeiten, weil die historischen Diagramme nur dazu dienen, die vorgenommenen Veränderungen zu veranschaulichen. Es wird angenommen, dass für das Diagramm A die historischen Diagramme A20150115203745, A20150116120711, A20150120120124 gibt. Wenn das Diagramm A20150116120711 geöffnet ist, werden die Unterschiede zwischen den Diagrammen A20150116120711 und A20150115203745 nicht nur textuell in einem Informationskästchen sondern auch graphisch durch auf den veränderten Elementen überlappende rote Rechtecke gekennzeichnet. Dies ist das *statische Wiedergeben von Veränderungen* im WebVbMF und basiert auf dem obengenannten Komparator.

Durch Doppelklick auf das Informationskästchen im historischen Diagramm können die Veränderungen dieses Diagramms dynamisch gezeigt werden. Beispielsweise wird für das Diagramm A zuerst das Diagramm A20150115203745 gezeigt, 3 Sekunden später sind die Unterschiede zwischen A20150116120711 und A20150115203745 herausgefunden und gezeigt, dann werden wieder in 3 Sekunden später die Unterschiede zwischen A20150120120124 und A20150116120711 gezeigt. Dies ist das *dynamische Wiedergeben von Veränderungen* im WebVbMF, es basiert ebenfalls auf dem Komparator.

Ein serverseitiges Servlet, der *HistoryShower*, ist für das dynamische Wiedergeben von Veränderungen verantwortlich. Die Aufgaben von HistoryShower sind, alle historischen Dateien eines Diagramm zu finden, nach der zeitlichen Reihenfolge die benachbarten Diagramme zu vergleichen, sowie das entsprechende Javascript, das die Veränderungen eines Diagramm dynamisch wiedergeben kann, zu generieren. Das Javascript, das durch den HistoryShower generiert ist, weist immer die im Listing 4

zeigte Form auf.

```
addComponent("Component","WQfbwbd2DR",150,80,169,231,"gTU3KvGKmE");
addAnnotation("Annotation","un5gvgO5Jz",110,40,424,501,"kFprVoT13f");
addNormalConnection("VmRTb4Gv9J","kFprVoT13f","gTU3KvGKmE");
var Counter = -1;
var Iteractor = setInterval(function(){
     Counter++;
     if (Counter==0){
          Round0();
     }
     if (Counter==1){
          Round1();
     }
     if (Counter==2){
          Round2();
     }
     if (Counter==3){
          Round3();
     }
}, 3000);
function Round3(){alert("Done!");}
function Round0(){
     applyPositionChange("un5gvgO5Jz",169,69);
}
function Round1(){
     deleteNormalConnection("kFprVoT13f","gTU3KvGKmE");
}
function Round2(){
     Workflow.getCommandStack().execute(new addInputPort("WQfbwbd2DR","cm",
          "flVRlx4Rbl",93,-8,80,20,291,147));
}
```

Listing 4: Durch den HistoryShower generiertes JavaScript

Dieses JavaScript wird als Bestandteil der HTTP-Antwort dem Client übertragen und dort aufgeführt. Als Ergebnis werden die Veränderungen dynamisch angezeigt.

5.6 Persistenz

Eclipse besitzt einen vollständigen Mechanismus zur Persistierung der Arbeitsumgebung. Die Größe des Editors, die geöffneten Dokumente, die Positionen jeder UI-Komponente usw. können beim Öffnen von Eclipse wiederhergestellt werden. Wenn Eclipse geschlossen wird, werden mithilfe des *XMLMemento* die Stati der UI-Komponenten, wie Workbench, Sichten, Editoren usw. gesammelt und in ein DOM integriert, dann wird das ganze DOM in ein XML-Dokument gespeichert. Beim Öffnen

von Eclipse werden die im XML-Dokument gespeicherten Informationen über die Status der UI-Komponenten durch *IMemento* übersetzt, anhand der übersetzten Informationen können die gespeicherten Status der UI-Komponenten wiederhergestellt werden. Als Plugin kann WebVbMF auch diesen Mechanismus benutzen.

Im WebVbMF sind die Positionen, die Größen aller UI-Komponenten fixiert, es gilt nur den Editor wieder herzustellen. Deshalb ist die Wiederherstellung von Eclipse für WebVbMF zu kompliziert und nicht notwendig. Darüber hinaus kann dieser Mechanismus problematisch sein, da er grundsätzlich für lokale Anwendungen entworfen ist, das heißt, dass nur die Einstellung eines Benutzers wiederhergestellt werden. Dagegen müssen bei WebVbMF die Einstellungen mehren Benutzer wiederhergestellt werden. Als Ergebnis muss im WebVbMF ein eigener Mechanismus zur Persistierung des Editors entwickelt wird. Die Persistenz im WebVbMF funktioniert wie folgt.

Um die Persistenz zu ermöglichen müssen die Projektnamen und Diagrammnamen aller geöffneten Diagramme jedes Benutzers gespeichert werden. Dies wird mithilfe einer Datenbank erreicht. Wenn ein Benutzer ein Diagramm öffnet, wird einen neuer Datensatz generiert und gespeichert. Wenn er dieses Diagramm schließt, wird dieser Datensatz gelöst. Beim Einloggen eines Benutzers werden die mit diesem Benutzer verbundenen Datensätze über die geöffneten Diagramme herausgefunden. Falls die entsprechenden Diagramme nicht gesperrt ist, werden sie dann automatisch im Editor geöffnet. Diese Aufgabe wird nach der Initialisierung des Systems erledigt. Aber die Frage ist, wie die Statusveränderungen des Editors überwacht werden können?

Der Überwacher für die asynchrone Kollaboration kann die Statusänderungen (geöffnet oder geschlossen) alle Diagramme überwachen. Deshalb ist es ideal, diesen Überwacher auch für die Persistenz zu verwenden. Aber aufgrund der Webbasiertheit ist der Überwacher für die Persistenz nicht geeignet, da die manuelle Aktualisierung (Schließen) der Webseite von WebVbMF mit dem Neustart (Beenden) von Eclipse äquivalent ist. Wenn Eclipse beendet wird, werden zuerst alle geöffneten Dokumente

geschlossen. Für ein webbasiertes Plugin Projekt von Eclipse gilt, wenn die Webseite des Plugin Projektes aktualisiert oder geschlossen wird, werden die im Editor geöffneten Dokumente vomm System geschlossen. Deshalb gilt wenn die Webseite von WebVbMF aktualisiert oder geschlossen werden, werden zuerst alle geöffneten Diagramme vom System geschlossen und die entsprechenden Datensätze über die geöffneten Diagramme automatisch gelöscht. Als Ergebnis wird nach der Aktualisierung der Webseite durch WebVbMF oder beim nächsten Einloggen eines Benutzers kein Diagramm automatisch geöffnet werden. Es soll somit ein neuer Überwacher für die Persistenz, der *Editorverwalter*, entwickelt werden.

Wenn ein Benutzer ein Diagramm öffnet, erzeugt der Editorverwalter einen Datensatz, der den Namen des Benutzers, den Projektname und den Diagrammnamen des geöffneten Diagramms enthält. Der Schwerpunkt der Impelemetierung des Editorverwalters liegt aber darin, wie der entsprechende Datensatz beim Schließen eines Diagramms zu behandeln ist. Aufgrund der Tatsache, dass ein Diagramm dann geschlossen wird, wenn der Benutzer auf die „Schließen“-Schaltfläche eines Diagramm klickt, kann durch die Modifizierung der zugrundeliegenden Eclipse oder RAP-Klasse von, die die Verhalten des Editors bei der Schließung definiert, das Löschen der Datensätze durch die Persistenz durchgeführt werden. Die Methode makePartButtonActive() der RAP-Klasse ConfigurableStack ist genau die Methode, die übergeschrieben werden soll. Nach dem Überschreiben wird der entsprechende Datensatz vor dem Schließen eines Diagramms gelöscht. Als Ergebnis kann der Editorverwalter den entsprechenden Datensatz löschen, wenn ein Benutzer ein Diagramm schließt.

Zwischen Persistenz und Kollaboration können Konflikte entstehen. Das ist dann der Fall, wenn beispielsweise ein Diagramm aufgrund des Persistenzmechanismus beim Einloggen eines Benutzers automatisch geöffnet werden muss, aber es schon von anderem Benutzer während der asynchronen Kollaboration geöffnet wurde. In diesem Fall kann das Diagramm nicht geöffnet werden, da es durch die asynchrone Kollaboration gesperrt ist. Eine mögliche Lösung ist den verbundenen Datensatz für die

Persistenz unverändert beizubehalten und das Diagramm beim nächsten Einloggen desselben Benutzers automatisch zu öffnen, falls es geöffnet werden kann. Diese Lösung ist aber problematisch, da der Benutzer darüber nicht weiß, was beim Einloggen automatisch geöffnet wird. Deshalb ist im WebVbMF eine andere mögliche Lösung umgesetzt. Wenn ein Diagramm beim Einloggen eines Benutzers gesperrt ist, wird der verbundene Datensatz für die Persistenz gelöst. Obwohl diese Lösung die Persistenzeigenschaften des Systems beeinträchtigt, ist sie verständlicher als die erste Lösung. Zwischen der synchronen Kollaboration und der Persistenz gibt es keinen Konflikt, da kein Diagramm gesperrt wird.

5.7 Zugriffskontrolle

Wie in der Tabelle 4 im Abschnitt 3.5 gezeigt wird können unterschiedliche Arten von Stakeholdern Interesse an den C&C-Diagrammen haben. Manche Stakeholder haber aber keinen technischen Hintergrund und können daher die Diagramme beschädigen, das Verhalten solcher Stakeholder muss beschränkt werden. Darüber hinaus müssen bestimmte Regeln für die reibungslose Zusammenarbeit entworfen werden. Um diese Aufgaben zu erledigen muss ein Subsystem, also die Zugriffskontrolle, implementiert werden.

Die Zugriffskontrolle ist regelbasiert. Eine Regel besteht aus einem Subjekt, einem Objekt und einer Operation und besagt, dass ein Subjekt eine Operation auf einem Objekt ausführen darf. Ein Subjekt kann eine Person oder ein Programm sein. Ein Objekt kann dagegen eine beliebige Datenentität (z.B. Datei) oder Systemressource wie beispielsweise Drucker sein. Anhand von Punkten, auf die sich die Zugriffsrechte-Entscheidungen beziehen, lassen sich die Zugriffskontrollen in drei Modelle, also Discretionary Access Control (DAC), Mandatory Access Control (MAC) und Role Based Access Control (RBAC), unterteilen [40].

In einem DAC-Modell hat der Inhaber eines Objekts das Recht, selbst zu bestimmen, welches Subjekt das Zugriffsrecht auf dieses Objekt hat und auf welche Art und Weise

auf dieses Objekt zugegriffen werden soll. Die Informationen über die Zugriffsrechte werden mit der sogenannten Access Control Matrix (ACM) verwaltet und werden normalerweise in Access Control Lists (ACLs) oder „Capacity Lists“ gespeichert. Eine ACL repräsentiert eine Spalte einer ACM und wird mit einem Objekt verbunden, eine Capacity List repräsentiert dagegen eine Zeile einer ACM und wird mit einem Subjekt verbunden [40]. Folgende Tabelle zeigt eine mögliche ACM für die Dateien im WebVbMF.

ACL

Objekt / Subjekt	Private Dateien	Kollaborative Dateien
Inhalber	Lesen, Schreiben	Lesen, Schreiben
Teammitglied (Creater)	-	Lesen, Schreiben
Teammitglied (Watcher)	-	Lesen

Capacity Lists

Tabelle 5: Eine mögliche ACM im WebVbMF

Die privaten Dateien sind nur für den Inhaber zugänglich. Wenn der Inhaber ein Diagramm in eine Gruppe teilt, können der Inhaber und die Mitglieder dieser Gruppe, falls sie als *Creater* von Teamleiter in die Gruppe aufgenommen worden, die Dateien dieses Diagramms lesen und schreiben, dagegen haben die Mitglieder dieser Gruppe, die vom Teamleiter als *Watcher* in die Gruppe aufgenommen worden, nur lesenden Zugriff auf die DiagrammDateien . Das DAC-Modell ist geeignet, die Zugriffsrechte für die Dateien zu kontrollieren, aber wenn man einen komplizierten Prozess kontrollieren möchte, ist DAC nicht dazu in der Lage.

Die MAC zielt darauf ab, Informationsflüsse zwischen den Objekten und den Subjekten zu kontrollieren. In einem MAC-Modell wird jedes Objekt und jedes Subjekt eine Sicherheitsstufe zugewiesen. Ein sogenanntes Dominanzverhältnis bestimmt, ob die Informationen einer Sicherheitsstufe in eine andere Sicherheitsstufe einfließen darf. Das Bell-LaPadula-Modell[27] ist eine typisches MAC. In diesem Modell gibt es die zwei Dominanzverhältnisse No-Read-Up und No-Write-Down [4]. Da im WebVbMF kein Informationsfluss kontrolliert werden soll, wird MAC nicht verwendet.

[27] Es wurde von David Elliott Bell und Leonard J. LaPadula im Jahr 1973 entwickelt.

Der zentrale Begriff bei RBAC ist Rolle. Rollen können von individuellen Benutzern abstrahiert werden, und entsprechen der Tätigkeitsprofilen dieser Benutzer. Die Zugangsrechte werden an Rollen, nicht aber an Benutzer, vergeben. Die Rollen werden dann den Benutzern zugewiesen. Dadurch wird garantiert, dass mit steigender Zahl von Benutzern das Zugriffskontrollsystem noch übersichtlich und nicht fehlerträchtig ist. Das RBAC-Modell unterstützt direkt die Prinzipien der geringste Rechten und Aufteilung von Aufgaben und hat die Fähigkeit, sehr komplexen Prozess zu kontrollieren [40]. Im WebVbMF wird RBAC für die Zugriffskontrolle verwendet.

Der Rollenentwurf ist die wichtigste Aufgabe bei RBAC. Folgende Tabelle zeigt die Rollen, die erlaubten Aktivitäten jeder Rolle, und wie die Rollen vergeben werden.

Rolle	**Übergeordnete Rolle**	**Rechte**	**Zuweisung**
Eingeloggter Benutzer	Keine	Ausloggen. Neues Diagramm erstellen. Neue Gruppe erstellen. Alle Rechte von nicht eingeloggtem Benutzer.	Einloggen.
Nicht eingeloggter Benutzer	Keine	Neues Konto registrieren. Das System probieren. Das Hilfssystem benutzen.	Nicht einloggen.
Teamleiter	Eingeloggter Benutzer	Neues Mitglied aufnehmen. Das Recht des Mitgliedes ändern. Bestehendes Mitglied löschen. Die Form der Synchronisation ändern. Das Team löschen.	Erstellen ein Team.
Teammitglied	Eingeloggter Benutzer	Die Diagramme des Teams öffnen. Die Historie des Teams sehen.	In ein Team aufgenommen.
Inhalber	Teammitglied	Eigene Diagramme, die im Team mitbenutzt sind, öffnen, bearbeiten, löschen, umbenennen und privatisieren. Eigene Diagramme im Team mitbenutzen lassen.	Erstellen ein Diagramm.
Creater	Teammitglied	Die Diagramme des Teams bearbeiten.	Vom Teamleiter zugewiesen.

Rolle	Übergeordnete Rolle	Rechte	Zuweisung
Watcher	Teammitglied	-	Vom Teamleiter zugewiesen.

Tabelle 6: RBAC-Rollen bei WebVbMF

Wie die Tabelle zeigt können die RBAC-Rollen eine bestimmte Struktur aufweisen, das heißt, eine Rolle kann eine übergeordnete oder untergeordnete Rolle haben. Wenn eine Rolle einer anderen Rolle untergeordnet sie, kann sie die Zugriffsrechte der übergeordneten Rolle erben. Darüber hinaus kann ein Subjekt gleichzeitig viele Rolle spielen, beispielsweise Wenn ein Benutzer WebVbMF ein Team bildet und ein eigenes Diagramm in diesem Team teilt, hat er gleichzeitig die Rollen Inhalber und Creater. Es kann sein, dass manche Rollen nicht gleichzeitig vom selben Subjekt übernommen werden können. Das heißt sie schließen sich gegenseitig aus. Beispielsweise kann ein Benutzer nicht gleichzeitig eingeloggter Benutzer und nicht eingeloggter Benutzer sein. Es kann sein, dass ein Subjekt wenn es eine Rolle spielt es auch eine andere Rolle spielen muss. Das nennt man Role-Binding, z.B, wenn ein Benutzer Teamleiter ist, wird er gleichzeitig auch Creater in dieser Gruppe [40].

In diesem Kapitel wurde die WebVbMF Implementierung vorgestellt. Es ist klar, dass es möglich ist einen C&C-Editor webbasiert und kollaborativ zu gestalten. Durch einfache Veränderungen und Ergänzungen kann WebVbMF den Benutzern die Möglichkeit bieten, alle Arten von Diagrammen, die für die Softwarearchitekturdokumentation notwendig sind, im Internet/Intranet kollaborativ zu bearbeiten. Deshalb kann WebVbMF als Skelett für eine webbasierte und kollaborative Anwendung zur Softwarearchitekturdokumentation in Diagrammform betrachtet werden.

6 Fazit

Die Softwarearchitektur ist eine Sammlung der für das logische Denken des Softwaresystems wichtigen Softwarestrukturen, die aus Softwareelementen und deren Beziehungen bestehen. Die Softwarearchitektur spielt nicht nur beim dem Entwurf sondern auch bei der Implementierung eines Softwaresystems eine wichtige Rolle, weil sie nicht nur die Basis für die Organisationsstruktur, die Schätzungen für die Kosten und den Zeitplan, die Kommunikationen zwischen den Stakeholdern und das Training ist, sondern auch durch die Begrenzung des Gestaltungsspielraum bei Entwurf und Implementierung die Unsicherheit des Projektes senkt.

Der Wert einer Softwarearchitektur geht verloren, falls sie nicht wiederverwendet werden kann. Die Voraussetzung für die Wiederverwendung ist die Dokumentation. Die Softwarearchitektur kann mittels natürlichen Sprachen, Bildern, formaler Sprachen, ADLs oder Diagrammen dokumentiert werden. Die Softwarearchitektur ist eine komplexe Entität, deshalb ist es notwendig, die Aufmerksamkeit auf bestimmte Seiten der Softwarearchitektur zu beschränken, dadurch werden unterschiedliche Sichten abgeleitet, die Dokumentation der Softwarearchitektur beschreibt solche Sichten. Die wichtigen Sichten, die die architektonischen Strukturen widerspiegeln, lassen sich in Modulsicht, Component-And-Connector Sicht und Allokationssicht unterteilen. Die Architekturmodelle wie das 4+1 Sichtenmodell und Garland And Anthony's Viewpoints können dazu beitragen, die zu dokumentierenden Sichten zu bestimmen. Die zu dokumentierenden Sichten können auch ohne die Architekturmodelle durch die Analyse der Interessen der Stakeholder an der Softwarearchitektur herausgefunden werden. Sobald die zu dokumentierenden Sichten bestimmt sind, können sie mithilfe von Architektur-Tools dokumentiert werden. Bei der Dokumentation können Architekturstile hilfreich sein. Ein Architekturstil ist eine benannte Spezialisierung von Elementen, Beziehungstypen sowie einer Reihe von Constraints, jeder Stil ist aber nur für eine bestimmte Sicht anwendbar.

Wenn die Softwarearchitektur als Diagramme dokumentiert werden muss, müssen bei der Dokumentation graphische Architektur-Tools verwendet werden., Solche Tools können UML-basierten Anwendungen sein, die Anwendungen, mittels der nicht standardisierter Diagramme erzeugen oder Anwendungen, die speziell für die Dokumentation von bestimmten Softwarearchitekturen entwickelt sind. Manchmal ist es notwendig, eine Menge graphischer Architektur-Tools zu verwenden, um die gesamte Architektur in Diagrammform zu dokumentieren.

In dieser Arbeit wurde auch die Implementierung des eigenen graphischen Architektur-Tool WebVbMF, vorgestellt. Das WebVbMF liefert einen webbasierten und kollaborativen Editor zur Dokumentation der C&C-Sichten. Es basiert auf den High-level Service Component View von VbMF und kann für das Erstellen und das Bearbeiten von C&C-Diagrammen genutzt werden.

Momentan sind die Fähigkeiten von WebVbMF beschränkt. Es kann nur C&C-Diagramme erstellen und bearbeiten. Durch leichte Veränderungen und Ergänzungen könnten alle architektonischen Strukturen mit WebVbMF webbasiert und kollaborativ bearbeitet werden.

Anhang

Bedienungsanleitung

WebVbMF besteht aus vielen Subsystemen, die unterschiedliche Funktionen bieten. Folgende Tabelle zeigt alle wichtigen Funktionen von WebVbMF.

Name	Symbol	Erklärung
Einloggen		-
Ausloggen		-
Registrieren		Neues Konto erstellen. Die Voraussetzungen für die erfolgreiche Registrierung sind eine gültige und einzigartige E-Mail Adresse sowie ein einzigartiger Alias. Nach der Registrierung können die E-Mail Adresse und der Alias nicht verändert werden.
Kontomanagement		Die mögliche Aktion dieses Subsystems ist Ändern des Kennworts.
Teammanagement		Das Subsystem Teammanagement.
Aktualisieren		Die Projekte und deren Diagramme in der Projektsicht sowie der Verlauf aller Diagramme soll durch die Verwendung dieser Funktionen aktualisiert werden. Zu aktualisierend sind z. B. die Status der Diagramme (gesperrt oder freigegeben), der neue Eintrag in der Verlaufssicht.
Neues Diagramm		Mit diesem Assistent kann ein neues Diagramm schrittweise erstellt werden.
Projektmanagement		Das Subsystem Projektmanagement.
Hochladen		Die Dateien eines Diagramms auf dem Client können mit dieser Funktion hochgeladen werden. Da WebVbMF gleichzeitig die Dateien von VbMF und die im WebVbMF definierte Diagram-Datei unterstützt, können nicht nur Component-Datei und Componentx-Datei, sondern auch Diagram-Dateien hochgeladen und in WebVbMF geöffnet werden.
Herunterladen		Die Dateien eines Diagramms auf dem Server können mithilfe dieser Funktion heruntergeladen werden. Aber nur das im Editor geöffnete und aktive Diagramm kann heruntergeladen werden.

Name	Symbol	Erklärung
Reinigen		Diese Funktion ist nur für die geteilten Diagramme sinnvoll. Wenn ein mitbenutzte Diagramm schon gelöscht und dessen Verlauf für die Teammitglieder nicht mehr interessant ist, kann mithilfe dieser Funktion der Verlauf dieses Diagramms gelöscht werden, um die Verlaufssicht übersichtlich zu machen.
Ordnen		Ordnen der Einträge in der Verlaufssicht nach dem Zeitpunkten der Veränderungen.
Ordnen		Ordnen der Einträge in der Verlaufssicht nach dem Projektnamen.
Ordnen		Ordnen der Einträge in der Verlaufssicht nach dem Teamnamen.
Ordnen		Ordnen der Einträge in der Verlaufssicht nach dem Alias der Bearbeiter.
Filter		Diese Funktion liefert die Möglichkeit, die Einträge im Verlauf zu filtern.
Probieren		Probieren des WebVbMF. Das mit dieser Funktion herstellte Diagramm kann nicht gespeichert werden.
Hilfesystem		Öffnen des WebVbMF-Hilfessystem.

Tabelle 7: Die Funktionen von WebVbMF

Da die Subsysteme von Teammanagement und Projektmanagement komplexer als andere Subsysteme sind, werden sie im Folgenden noch detailliert vorgestellt.

Das Teammanagement beschäftigt sich mit der Verwaltung von Teams, was die Grundlage für die Kollaboration ist. Im Teammanagement kann jeder eingeloggte Benutzer eine unbegrenzte Anzahl von Teams bilden, bei dem Bilden eines Teams müssen der Name des Teams, die Beschreibung eingetragen werden, darüber hinaus muss über die Kollaborationsmethode des Teams entschieden werden. Die Kollaborationsmethode kann aber von dem Teamleiter jederzeit geändert werden. Wenn *Single User Mode* aktiviert ist, darf jedes im Team geteilte Diagramm gleichzeitig nur von einem Mitglied geöffnet werden, wenn dagegen *Multi User Mode* aktiviert ist, können alle Mitglieder dieses Teams gleichzeitig ein Diagramm öffnen und kollaborativ bearbeiten.

Durch Doppelklick auf die in der Tabelle *Team Information* gelisteten Teams kann der Teamleiter ein Dialogfenster öffnen, in dem er das entsprechende Team konfigurieren

kann. Dieses Dialogfenster liefert dem Teamleiter die Möglichkeiten, die Beschreibung des Teams und die Methode der Kollaboration in diesem Team zu ändern, die Rechte der Mitglieder zu bearbeiten, die Mitglieder zu löschen, neues Mitglieder ins Team aufzunehmen, sowie das Team zu löschen. Ein Mitglied kann *Creater* oder *Watcher* sein. Wenn ein Mitglied die Creater-Rolle von hat, darf er die im Team mitbenutzten Diagramme bearbeiten, eigene Diagramme im Team mitbenutzen lassen sowie das Teilen von Diagramme beenden. Wenn ein Mitglied ein Watcher ist, darf er nur die Diagramme des Teams sehen. Die Rollen der Mitglieder können von dem Teamleiter jederzeit verändert werden. Bei der Löschung eines Teams werden die Diagramme des Teams nicht gelöscht, sondern anhand der Inhaber privatisiert.

Das Projektmanagement kümmert sich um die Verwaltung der Diagramme. Jeder eingeloggte Benutzer kann diese Funktion nutzen. Wenn ein Benutzer das Dialogfenster Projektmanagement öffnet, werden zwei Tabellen angezeigt. Eine listet die Diagramme, die von diesem Benutzer erstellt werden, die andere zeigt die für diesen Benutzer zugänglich Diagramme der Teams. Der Benutzer hat das Recht, eigene Diagramme zu umbenennen, zu teilen, sowie zu löschen. Falls nach der Lösung eines Diagramms das Projekt leer ist, wird das Projekt auch gelöscht. Wenn ein Diagramm schon in einem Team geteilt wird, kann der Benutzer dieses Diagramm wieder exklusiv nutzen oder in einem anderen Team mitbenutzen lassen.

Nach dem erfolgreichen Einloggen eines Benutzers werden die mit diesem Benutzer verbundenen Diagramme baumartig in der Projektsicht gelistet. Wenn der Name eines Diagramms rot ist, wird dieses Diagramm im Team geteilt. Durch Doppelklick auf dem Name des Diagramms kann der Benutzer das Diagramm im Editor öffnen, falls das Diagramm nicht gesperrt ist.

Wenn ein Benutzer ein Diagramm im Team teilt, sowie wenn ein Benutzer ein mitbenutztes Diagramm modifiziert oder löscht, wird ein Eintrag in der Verlaufssicht generiert. Durch Doppelklick auf einem Eintrag kann das entsprechende historische Diagramm geöffnet werden. Alle historischen Diagramme können nicht modifiziert

werden. Sie dienen lediglich dazu, die Veränderungen zu veranschaulichen. Durch Doppelklick auf das Informationskästchen in einem historischen Diagramm kann der gesamte Verlauf dieses Diagramms dynamisch präsentiert werden.

Im Folgenden werden der Editor vorgestellt. Die Funktionen des Editors werden grundsätzlich in der Palette integriert. Folgende Tabelle zeigt die Funktionen von Editor.

Name	Symbol	Erklärung
Rückgängig		Eine Aktion rückgängig machen.
Wiederholen		Eine Aktion wiederholen.
Auswahlmodus		Den Auswahlmodus aktivieren oder deaktivieren. Wenn der Auswahlmodus aktiviert ist, wird das ausgewälte Element und seine Verbindungen herausstellend gezeigt.
Befestigen		Die Palette befestigen oder freigeben.
Komponente		Eine neue Komponente erzeugen.
Eingangsport		Einen Eingangsport erzeugen. Die Voraussetzungen dafür sind, dass eine Komponente schon ausgewählt ist und die Komponente noch freien Platz für den Eingangsport hat.
Ausgangsport		Einen Ausgangsport erzeugen. Die Voraussetzungen dafür sind, dass eine Komponente schon ausgewählt ist und die Komponente noch freien Platz für den Ausgangsport hat.
Property		Ein neues Property erzeugen.
Stereotyp		Einen neuen Stereotyp erzeugen.
Annotation		Eine neue Annotation erzeugen.
View Annotation		Eine neue View Annotation erzeugen.
Speichern		Das Diagramm an den Server übertragen und auf den Server speichern. Während des Probierens des Systems und für die Watchers ist diese Funktion nicht verfügbar.

Tabelle 8: Die Funktionen des Editors

Manche Funktionen können nicht nur in der Palette sondern auch im Kontextmenü aufgerufen werden. Angesichts des Abmessungen ist für jede Komponente eine Kapazität festgestellt, das heißt, dass eine Komponente nur eine bestimmte Anzahl von Ports hat. Wenn eine Komponente auslastet ist, kann durch Vergrößerung der Komponente zusätzliche Plätze für die Ports bekommen.

Eine Verbindung kann nicht mithilfe der Palette erzeugt werden, sie wird durch

Drag&Drop des Ausgangsport, falls das Ziel des Drop ein Eingangsport ist, erzeugt. Die Palette bietet nicht die Möglichkeit ein Element zu löschen. Das Löschen kann via Löschtaste oder Kontextmenü durchgeführt werden.

Literaturverzeichnis

[1] Anlauff, M. und Sünbül, A. (2000): Domain-Specific Languages in Software Architecture - Integrated Design And Process Technology. In: IEEE International Conference on Systems Integration, Dallas. Erhältlich unter: www.kestrel.edu/home/people/ma/papers/DSLSA.pdf, letzter Abruf: 04.02.2015.

[2] Baike.baidu.com (2014): ATAM. http://baike.baidu.com/view/1525020.htm, letzter Abruf: 21.10.2014.

[3] Bass, L. et al. (2012): Software Architecture in Practice. Addison-Wesley Professional, Indianapolis, 3. Auflage.

[4] Benantar, M. (2006): Access Control Systems - Security, Identity Management and Trust Models. Springer Science & Business Media, New York. S. 129-146.

[5] Borghoff, M. U. und Schlichter, H. J. (2000): Computer-Supported Cooperative Work - Introduction to Distributed Applications. Springer-Verlag, Berlin. S. 172-177.

[6] Buyya, R. et al. (2013): Mastering Cloud Computing. McGraw-Hill Education, New Delhi. 2.11-2.19.

[7] Clements, P. et al. (2011): Documenting Software Architectures - Views and Beyond. Pearson Education, Boston, 2. Auflage. S. 3-329.

[8] DOM4J (2015): DOM4J. dom4j.sourceforge.net/, letzter Abruf: 15.01.2015.

[9] Ellis, C. A. und Gibbs, S. J. (1989): Concurrency control in groupware systems. In: SIGMOD '89 Proceedings of the 1989 ACM SIGMOD international conference on Management of data. ACM, New York. S. 399-407.

[10] Ferrate, A. (2010): Google Wave - Up and Running. O'Reilly Media, Sebastopol,

First Edition. S. 27.

[11] Flanagan, D. (2007): JavaScript - das umfassende Referenzwerk. O'Reilly Verlag, Köln, 5. Auflage. S. 1-3.

[12] Flügge, M. (2009): Diplomarbeit - Entwicklung einer kollaborativen Erweiterung für GMF-Editoren auf Basis modellgetriebener und webbasierter Technologien. Erhältlich unter: www.mftech.org/dawn/documents/Diplomarbeit_Martin_Fluegge.pdf, letzter Abruf: 10.12.2014.

[13] Garland, J. und Anthony, R. (2003): Large-Scale Software Architecture - A Practical Guide using UML. John Wiley & Sons, West Sussex. S. 1-38, 69-200.

[14] Gronback, R.C. (2009): Eclipse Modeling Project - A Domain-Specific Language (DSL) Toolkit. Pearson Education, Boston. S. 317-325.

[15] IBM (2006): Which one is better, SWT, Swing or AWT? https://www.ibm.com/developerworks/cn/opensource/os-swingswt/, letzter Abruf: 06.12.2014.

[16] IBM (2014): Rational Software Architect. http://www-03.ibm.com/software/products/en/ratisoftarch, letzter Abruf: 20.11.2014.

[17] INDENICA (2010-2013): About the Project. http://www.indenica.eu/index.php?id=403, letzter Abruf: 20.11.2014.

[18] INDENICA (2011): View-based Design Time and Runtime Architecture for TailoringVirtual Service Platforms. Erhältlich unter: www.indenica.eu/fileadmin/INDENICA/user_upload/d31-viewbasedarch.pdf, letzter Abruf: 20.11.2014. S. 6-19.

[19] Ivers, J. et al. (2004): Documenting Component and Connector Views with UML 2.0. Carnegie Mellon University, erhältlich unter: www.sei.cmu.edu/reports/04tr008.pdf, letzter Abruf: 21.01.2015.

[20] Jackson, P. und Moulinier, I. (2007): Natural Language Processing for Online

Applications - Text Retrieval, Extraction and Categorization. John Benjamins B. V., Amsterdam, Second revised edition. S. 1-20.

[21] JDOM (2015) : JDOM. http://www.jdom.org/, letzter Abruf: 15.01.2015.

[22] Kang, B. (2002): XML Programmiertechniken. Tsinghua University Press, Beijing. S. 201.

[23] Kogut, P. und Clements, P. (1995): Feature Analysis of Software Architecture Description Languages. Software Technology Conference, Salt Lake City, erhältlich unter: www.sei.cmu.edu/library/assets/features_of_adls.pdf, letzter Abruf: 06.11.2014.

[24] Kruchten, P. (1995): The 4+1 View Model of Software Architecture. www.ibm.com/developerworks/cn/rational/r-4p1-view/, letzter Abruf: 04.11.2014.

[25] Kruchten, P. et al. (2006): The Past, Present, and Future for Software Architecture. In: IEEE Software. Vol. 23, Nr. 2. S. 22-30.

[26] Lange, F. (2008): Eclipse Rich Ajax Platform - Bringing Rich Client to the Web. firstPress, New York. S. 12-19.

[27] Lee, W. und Foo, S. (2002): XML Programming Using the Microsoft XML Parser. Apress, Berkeley. S. 107-108.

[28] Li, G. (2006): Professional Java Native Interfaces with SWT/JFace. John Wiley & Sons. S. 3-18.

[29] Linz, P. (2011): An Introduction to Formal Languages and Automata. Jones & Bartlett Learning, Sudbury, Fifth edition. S. 1-13.

[30] Liu, C. (2008): Java Web Entwicklung - JSP+Servlet+Struts+Spring+Hibernate+Ajax. China Machine Press, Beijing. S. 78.

[31] Lytra, I. et al. (2013): Supporting Consistency between Architectural Design Decisions and Component Models through Reusable Architectural Knowledge

Transformations. In: Software Architecture - 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings. Springer-Verlag, Berlin. S. 227.

[32] Marechaux, J. (2012): Using Rational Software Architect To Define Software Architecture. http://www.ibm.com/developerworks/cn/rational/define-application-architecture-rational-software-architect-1/index.html, letzter Abruf: 10.02.2015.

[33] Mittal, K. (2005): Introducing IBM Rational Software Architect - Improved usabilitymakes software development easier. http://www.ibm.com/developerworks/rational/library/05/kunal/index.html, letzter Abruf: 20.11.2014.

[34] Oestereich, B. und Scheithauer, A. (2012): Analyse und Design mit der UML 2.5 - Objektorientierte Softwareentwicklung. Oldenbourg Verlag, München, 10. Auflage. S. 21-23,241.

[35] Randell, B. und Buxton, J. (1970): Software Engineering Techniques - Report of aconference sponsored by the NATO Science Committee. Scientific Affairs Division, Brussels, erhältlich unter: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF, letzter Abruf: 19.10.2014. S. 9.

[36] Rozanski, N. und Woods, E. (2011): Software Systems Architecture - Working with Stakeholders Using Viewpoints and Perspectives. Pearson Education, Boston, 2. Auflage. S.9-62, 161-176, 241-424.

[37] Simsion, G. (2007): Data Modeling - Theory and Practice. Technics Publications, Bradley Beach. S. 23-62.

[38] Software Engineering Insititute Of Carnegie Mellon University (2014): CommunitySoftware Architecture Definitions. www.sei.cmu.edu/architecture/start/glossary/community.cfm, letzter Abruf: 12.10.2014.

[39] Streekmann, N. (2011): Clustering-Based Support for Software Architecture Restructuring. Vieweg+Treubner Verlag, Heidelberg. S. 9-22.

[40] Strembeck, M. und Mendling, J. (2011): Modeling process-related RBAC model with extended UML activity models. In: Information and Software Technology, Volume 53. Elsevier, London. S. 456-483.

[41] Sun, C. et al. (1998): Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. In: ACM Transactionson Computer-Human Interaction, Vol. 5, No. 1. ACM, New York. S. 63-108.

[42] SWA (2014): INDENICA View-based Modeling Framework. swa.univie.ac.at/INDENICA_View-based_Modeling_Framework, letzter Abruf: 12.02.2015.

[43] System Architect Conference China (SACC) (2014): System Architect Conference China 2104. http://sacc.it168.com/, letzter Abruf: 03.02.2015.

[44] The Eclipse Foundation (2014a): Remote Application Platform. http://projects.eclipse.org/projects/rt.rap, letzter Abruf: 24.10.2014.

[45] The Eclipse Foundation (2014b): Mission Statement. http://eclipse.org/equinox/, letzter Abruf: 25.10.2014.

[46] The Software Architecture Portal (1999): WICSA1: First Working IFIP Conference on Software Architecture. http://users.ece.utexas.edu/~perry/prof/wicsa1/, letzter Abruf: 08.02.2015.

[47] UMLChina (2014): Die Tools für die UML. www.umlchina.com/Tools/Newindex1.htm, letzter Abruf: 20.11.2014.

[48] W3C (2014): HTML5 - A vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/2014/REC-html5-20141028/, letzter Abruf: 23.11.2014.

[49] W3School (2014): HTML Tag <canvas>. http://www.w3school.com.cn/tags/tag_canvas.asp, letzter Abruf: 27.11.2014.

[50] Xue, L. et al. (2002): Intention Preservation by Multi-versioning in Distributed Real-Time Group Editors. In: Engineering and Deployment of Cooperative Information Systems - First International Conference, EDCIS 2002, Beijing, China, September 17-20, 2002. Proceedings. Springer-Verlag, Berlin. S. 510-524.

[51] Ye, W. (2009): Software Revolution In The Internet Age - SaaS Architecture. Publishing House Of Electronics Industry, Beijing. S. 50-57.

[52] yWorks (2014): yEd Graph Editor. http://www.yworks.com/en/products/yfiles/yed/, letzter Abruf: 20.11.2014.

[53] Zhang, P. et al. (2008): Eclipse Plugin Development. Publishing House Of Electronics Industy, Beijing. S. 12-14,19-21.

[54] Zheng, Q. et al. (2009): Software Architecture. Zhejiang University Press, Hangzhou und Springer-Verlag, Berlin Heidelberg. S. 1-88, 117-168.

[55] Zhu, H. (2005): Software Design Methodology - From Principles to Architectural Styles. Butterworth-Heinemann, Burlington. S. 73-172.