Entwicklung einer kollaborativen Erweiterung für GMF-Editoren auf Basis modellgetriebener und webbasierter Technologien

Martin Flügge

Hochschule für Technik und Wirtschaft
Fachbereich 4 – Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

Gutachter:
Prof. Dr. Elke Naumann
Prof. Dr. Hermann Heßling

Abstrakt


GMF, the Graphical Modeling Framework, offers methods to create graphical editors with the help of model-driven techniques. Those editors can be used in various ways. Yet, no technology has been developed which allows the collaborative usage of them. To provide such a platform Dawn, a collaborative, web based extension for GMF-editors, had been developed. Dawn provides shared editing for GMF-diagrams as well as methods to detect and avoid conflicts. Aside the communication infrastructure the system allows viewing GMF-diagrams in web-browsers. This functionality can be used to view such diagrams on mobile devices. Because mobile devices sometimes are used in not reliable network environments Dawn can operate without network connection. To avoid unauthorized access to the system, a generic user concept had been developed which can also operate without connection to the Dawn server.
Inhaltsverzeichnis

ABSTRAKT ................................................................................................................................. II

INHALTSVERZEICHNIS ........................................................................................................... III

1 EINLEITUNG ............................................................................................................................. 1
1.1 MOTIVATION ....................................................................................................................... 1
1.2 ZIELSETZUNG ..................................................................................................................... 2
1.3 AUFBAU DER ARBEIT ......................................................................................................... 3
1.4 KONVENTIONEN DER ARBEIT .......................................................................................... 5

2 GRUNDLAGEN ...................................................................................................................... 6
2.1 MODELLE ............................................................................................................................. 6
2.2 FORMALE MODELLE .......................................................................................................... 7
2.3 META-MODELLE .............................................................................................................. 8
2.4 META²-MODELL ............................................................................................................. 10
2.5 MDSD – MODEL DRIVEN SOFTWARE DEVELOPMENT .................................................. 11
2.5.1 Domänen und domänenspezifische Sprachen .......................................................... 12
2.5.2 Plattformen .................................................................................................................. 13
2.5.3 Modelle der MDSD ....................................................................................................... 13
2.5.3.1 CIM – Computation Independent Model ................................................................ 13
2.5.3.2 PIM – Platform Independent Model .......................................................................... 13
2.5.3.3 PSM – Platform Specific Model ............................................................................. 14
2.5.3.4 Modell-Transformationen ....................................................................................... 14
2.5.4 Generatoren ............................................................................................................. 15
2.5.5 Vor- und Nachteile der MDSD ..................................................................................... 15
2.5.6 Anverwandte Disziplinen ......................................................................................... 16
2.6 UML – UNIFIED MODELING LANGUAGE ........................................................................ 18
2.6.1 Diagrammwelten ...................................................................................................... 18
2.6.2 MOF – Meta Object Facility .................................................................................... 20
2.7 OSGI SERVICE PLATFORM ............................................................................................. 21
2.7.1 Einsatzgebiete .......................................................................................................... 21
2.7.2 OSGi-Framework ....................................................................................................... 22
2.7.3 Bundles ..................................................................................................................... 24
2.7.4 Services .................................................................................................................... 26
2.7.5 OSGi-Implementierungen ....................................................................................... 27
2.7.6 OSGi und SOA ......................................................................................................... 28
2.8 ECLIPSE ........................................................................................................................... 30
2.8.1 Die Plattform – Equinox .......................................................................................... 32
2.8.2 RCP – Rich Client Platform .................................................................................... 33
2.8.3 PDE – Plugin Development Environment ................................................................ 33
2.8.3.1 Plugins ................................................................................................................... 34
| 2.8.3.2 | Extensions | 35 |
| 2.8.3.3 | Extension Points | 36 |
| 2.8.3.4 | Fragmente | 38 |
| 2.8.3.5 | Ansichten, Editoren und Perspektiven | 38 |
| 2.8.4 | EMF – Eclipse Modeling Framework | 39 |
| 2.8.4.1 | EMF.Core – Ecore | 39 |
| 2.8.4.2 | Generator Modell | 40 |
| 2.8.4.3 | EMF.Edit und EMF.Codegen | 40 |
| 2.8.4.4 | Resource und ResourceSet | 41 |
| 2.8.4.5 | EMF und XMI | 41 |
| 2.8.4.6 | Dynamic EMF | 41 |
| 2.8.5 | Exkurs – SWT, JFace und Draw2D | 42 |
| 2.8.6 | GEF – Graphical Editing Framework | 43 |
| 2.8.6.1 | Requests, Commands und EditPolicies | 44 |
| 2.8.7 | GMF – Graphical Modeling Framework | 45 |
| 2.8.7.1 | GMF-Tooling | 46 |
| 2.8.7.2 | GMF-Runtime | 48 |
| 2.8.7.3 | Figure Deskriptoren | 49 |
| 2.8.7.4 | ElementTypes | 50 |
| 2.8.7.5 | Vor- und Nachteile von GMF | 50 |
| 2.9 | ERWEITERTE GRUNDLAGEN | 50 |
| 2.9.1 | Web Services | 51 |
| 2.9.2 | Apache Axis | 52 |
| 2.9.3 | Hibernate | 52 |
| 2.9.4 | Ajax | 52 |
| 2.10 | ZUSAMMENFASSUNG | 53 |

3 ANFORDERUNGSANALYSE UND -DEFINITION 54

3.1 BETRACHTUNG ECLIPSE-BASIERTER NETZWERK-FRAMEWORKS 54
3.1.1 CDO – Connected Distributed Objects 54
3.1.2 ECF – Eclipse Communication Framework 56

3.2 GENERATOR FRAMEWORKS 56
3.2.1 JET – Java Emitter Templates 57
3.2.2 XPand – oAW 57

3.3 WEB FRAMEWORKS 58

3.4 ANWENDUNGSSZENARIEN 61

3.5 SYSTEM ARCHITEKTUR 63

3.6 SYSTEMANFORDERUNGEN 64
3.6.1 Kommunikation und Firewall-Transparenz 64
3.6.2 Ausfallresistenz, Mobilität und Netzwerkunabhängigkeit 65
3.6.3 Web-Viewer und Systemverwaltung 65
3.6.4 Synchronisation und Ressourcenverwaltung 66
3.6.5 User-Management und Sicherheit 67
3.6.6 Generierung der Komponenten 67
3.6.7 Anforderungen an das prototypische Diagramm 67
3.6.8 Systemumgebung 68

3.7 WUNSCHKRITERIEN 68
## Inhaltsverzeichnis

<table>
<thead>
<tr>
<th>Kapitel</th>
<th>Titel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>ABGRENZUNGEN</td>
<td>69</td>
</tr>
<tr>
<td>3.9</td>
<td>ANFORDERUNGSKATALOG</td>
<td>69</td>
</tr>
<tr>
<td>4</td>
<td>KONZEPTENTWICKLUNG UND SYSTEMDESIGN</td>
<td>70</td>
</tr>
<tr>
<td>4.1</td>
<td>SYSTEMARCHITEKTUR</td>
<td>71</td>
</tr>
<tr>
<td>4.2</td>
<td>KOMMUNIKATIONSSTRUKTUR</td>
<td>74</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Netzwerkadapter</td>
<td>75</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Austausch der Ressource</td>
<td>76</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Watcher</td>
<td>77</td>
</tr>
<tr>
<td>4.3</td>
<td>SYNCHRONISATION</td>
<td>78</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Datenübertragung</td>
<td>79</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Operationen</td>
<td>81</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Compare Engine</td>
<td>82</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Merge Engine</td>
<td>83</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Schutz lokaler Änderungen</td>
<td>84</td>
</tr>
<tr>
<td>4.4</td>
<td>KONFLIKTE</td>
<td>85</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Konflikte – Client Seite</td>
<td>86</td>
</tr>
<tr>
<td>4.4.1.1</td>
<td>Lokal- und Remote-Änderungskonflikt</td>
<td>86</td>
</tr>
<tr>
<td>4.4.1.2</td>
<td>Lokaler Löschkonflikt</td>
<td>87</td>
</tr>
<tr>
<td>4.4.1.3</td>
<td>Entfernter Löschkonflikt</td>
<td>88</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Konflikte – Serverseite</td>
<td>89</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Konflikterkennung</td>
<td>93</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Konfliktbeseitigung</td>
<td>94</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Konfliktvermeidung</td>
<td>97</td>
</tr>
<tr>
<td>4.5</td>
<td>NETZWERKUNABHÄNGIGKEIT</td>
<td>98</td>
</tr>
<tr>
<td>4.6</td>
<td>AUTHENTIFIZIERUNG UND AUTORISIERUNG</td>
<td>101</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Rollen und Operationen</td>
<td>101</td>
</tr>
<tr>
<td>4.6.2</td>
<td>UserManager Konzept</td>
<td>102</td>
</tr>
<tr>
<td>4.6.3</td>
<td>GlobalUserManager und NullUserManager</td>
<td>104</td>
</tr>
<tr>
<td>4.7</td>
<td>DATENBANK UND PERSISTENZ</td>
<td>105</td>
</tr>
<tr>
<td>4.8</td>
<td>WEBKOMPONENTEN</td>
<td>107</td>
</tr>
<tr>
<td>4.8.1</td>
<td>Web-Viewer</td>
<td>107</td>
</tr>
<tr>
<td>4.8.1.1</td>
<td>Darstellung der Diagramme</td>
<td>108</td>
</tr>
<tr>
<td>4.8.1.2</td>
<td>Kommunikation</td>
<td>111</td>
</tr>
<tr>
<td>4.8.2</td>
<td>Web-Frontend</td>
<td>112</td>
</tr>
<tr>
<td>4.9</td>
<td>DAWN-RUNTIME UND EDITOR-ERWEITERUNG</td>
<td>114</td>
</tr>
<tr>
<td>4.9.1</td>
<td>Wizards</td>
<td>115</td>
</tr>
<tr>
<td>4.9.2</td>
<td>Preferences</td>
<td>115</td>
</tr>
<tr>
<td>4.9.3</td>
<td>Dawn Conflict View</td>
<td>115</td>
</tr>
<tr>
<td>4.10</td>
<td>KONZEPT DES PROTOTYPISCHEN DIAGRAMMS</td>
<td>116</td>
</tr>
<tr>
<td>4.11</td>
<td>CODE-GENERIERUNG</td>
<td>118</td>
</tr>
<tr>
<td>4.11.1</td>
<td>Dawn-GenModel</td>
<td>118</td>
</tr>
<tr>
<td>4.11.2</td>
<td>Generierung der Komponenten</td>
<td>119</td>
</tr>
<tr>
<td>4.12</td>
<td>ZUSAMMENFASSUNG</td>
<td>121</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTIERUNG</td>
<td>123</td>
</tr>
<tr>
<td>5.1</td>
<td>KOMMUNIKATION UND SYNCHRONISATION</td>
<td>123</td>
</tr>
</tbody>
</table>
ZUSAMMENFASSUNG.................................................................................................................. 168
7.1 ERGEBNISBETRACHTUNG............................................................... 168
7.2 SCHLUSSFOLGERUNGEN UND ERFAHRUNGEN................................. 170
7.3 ZUKünFTIGE ERWEITERUNGEN.................................................... 171

LITERATURVERZEICHNIS................................................................. IX
INTERNETQUELLEN ..................................................................... XIV
ABBILDUNGSVERZEICHNIS........................................................ XVIII
TABELLENVERZEICHNIS................................................................. XX
LISTINGS ....................................................................................... XX
GLOSSAR ...................................................................................... XXI
INDEX ........................................................................................ XXVIII
ANHANG ....................................................................................... XXX
A ANFORDERUNGSKATALOG.......................................................... XXX
<table>
<thead>
<tr>
<th>Kapitel</th>
<th>Titel</th>
<th>Seitenzahl</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Einleitung</td>
<td>XXXX</td>
</tr>
<tr>
<td>A.2</td>
<td>Zielbestimmungen</td>
<td>XXXY</td>
</tr>
<tr>
<td>A.2.1</td>
<td>Musskriterien</td>
<td>XXX</td>
</tr>
<tr>
<td>A.2.2</td>
<td>Wunschkriterien</td>
<td>XXXIII</td>
</tr>
<tr>
<td>A.2.3</td>
<td>Abgrenzungskriterien</td>
<td>XXXIV</td>
</tr>
<tr>
<td>A.3</td>
<td>Produkteinsatz</td>
<td>XXXV</td>
</tr>
<tr>
<td>A.4</td>
<td>Produktumgebung</td>
<td>XXXV</td>
</tr>
<tr>
<td>A.4.1</td>
<td>Software</td>
<td>XXXV</td>
</tr>
<tr>
<td>A.4.2</td>
<td>Hardware</td>
<td>XXXV</td>
</tr>
<tr>
<td>B</td>
<td>INHALT DER BEIGEFÜGTEN CD</td>
<td>XXXVI</td>
</tr>
</tbody>
</table>
1 Einleitung


1.1 Motivation

All diese Vorteile haben in den letzten Jahren sowohl auf dem kommerziellen Sektor, als auch im Open-Source Bereich dazu geführt, dass eine Vielzahl unterschiedlicher Software für die Entwicklungsunterstützung erstellt wurde.

Einleitung

Das *Graphical Modeling Framework (GMF*)\(^1\), ein Projekt der Eclipse Foundation, beschäftigt sich mit dem generativen Erstellen von grafischen Editoren. Hierbei ist es mittels modellgetriebener Softwareentwicklung möglich, grafische Editoren für die verschiedensten Anwendungsgebiete zu erstellen. *GMF* stellt sowohl einen Generator zum Erstellen der Editoren, als auch eine Laufzeitumgebung zur Benutzung dieser zur Verfügung.


### 1.2 Zielsetzung


\(^1\) https://www.google.com
Einleitung

Das System wird folglich aus einer Laufzeitkomponente, welche eine Kommunikations- und Synchronisationsplattform zu Verfügung stellt, und einer generativen Komponente bestehen, die für neu zu entwickelnde bzw. schon existierende Editoren die entsprechenden Kommunikationsadapter erstellen kann. Über eine einheitliche Datenbasis sollen die Projektinformationen jederzeit für alle Entwickler zur Verfügung stehen, sodass sich der Entwicklerkreis dynamisch vergrößern und verkleinern kann, ohne dass Projektinformationen verloren gehen.


1.3 Aufbau der Arbeit

Einleitung


Um den Aufgabenbereich definieren zu können und einsetzbare Technologien zu betrachten, werden im Kapitel Anforderungsanalyse und -definition die Anforderungen an das zu entwickelnde System analysiert. Dies geschieht unter dem Fokus einen Anforderungskatalog zu erstellen, welcher als Leitfaden für Entwicklung des Konzeptes und die Implementierung dienen soll. Das umfasst die Betrachtung bestehender Netzwerk- und Generator-Frameworks und deren Begutachtung hinsichtlich der Einsetzbarkeit innerhalb der Arbeit.


Die Beschreibung der Implementierung des Konzeptes behandelt im anschließenden Themenkomplex ausgewählte Umsetzungen. Hierbei werden vor allem jene Aspekte aufgegriffen, welche besonders schwierig oder interessant waren.

In der Ergebnisbetrachtung erfolgt eine Auswertung hinsichtlich der Umsetzung der definierten Ziele. Dabei werden die Ergebnisse kritisch betrachtet und ihr Nutzen herausgestellt. Zusätzlich werden die in dieser Arbeit gewonnenen Erfahrungen resümiert und ein Ausblick auf die zukünftige Entwicklung des Systems gegeben.

1.4 Konventionen der Arbeit

<table>
<thead>
<tr>
<th>Kursiv</th>
<th>Für Abkürzungen und Eigennamen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier New</td>
<td>Klassenbezeichner und Quellcode spezifische Bezeichner innerhalb des Fließtextes</td>
</tr>
<tr>
<td>Courier New und Courier New Fett</td>
<td>Für Quelltexte und Listing</td>
</tr>
<tr>
<td>“Times New Roman, kursiv”</td>
<td>Zitate</td>
</tr>
</tbody>
</table>
2 Grundlagen


2.1 Modelle

„A model captures a view of a physical system. It is an abstraction of the physical system, with a certain purpose. [...] Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail“ [Rupp et al. 2007, S.36].

Dieser Definition folgend stellen Modelle eine abstrakte Darstellung eines Systems dar, welche in Detaillierungsgrad und Ausprägung von der eigentlichen Zielbestimmung des
Modells abhängt. Hierbei versuchen Modelle die Komplexität vorhandener Systeme in einem Maße zu reduzieren, dass bezüglich einer konkreten Fragestellung die Aussage des Systems erhalten bleibt, für den Nutzer des Modells die Bearbeitung der Fragestellung aber wesentlich erleichtert wird.

Vereinfacht gesagt filtern Modelle Eigenschaften von komplexen Systemen hinsichtlich eines Problems und stellen diese in geeigneter Form dar. Es erfolgt also eine Reduktion der Komplexität auf die für die Problemstellung notwendigen Systemeigenschaften.


Diese Beispiele sollen verdeutlichen, dass Modelle nicht an eine bestimmte Form gebunden sind. Sie können sowohl in textueller, grafischer, plastischer, gedanklicher als auch kombinierter Form vorliegen.

### 2.2 Formale Modelle

Modelle finden also in verschiedensten wissenschaftlichen Disziplinen ihre Anwendung. Doch nicht jedes Modell lässt sich für die Softwareentwicklung nutzen. Um innerhalb von modellgetriebener Softwareentwicklung anwendbar zu sein, muss ein Modell einen Aspekt vollständig beschreiben. Ein Modell, welches diese Anforderung erfüllt, nennt sich *formal* [vgl. Stahl et al. 2007, S.11 ff.].

Diese Regeln werden genutzt, um den Modellinhalt derart zu spezifizieren, dass er mit Hilfe von Transformatoren in Quellcode überführt werden kann. Dieser Quellcode muss nicht, kann aber lauffähig sein.


### 2.3 Meta-Modelle


Das Meta-Modell spezifiziert hierbei die abstrakte Syntax einer modellbeschreibenden Sprache. Diese definiert die Strukturen, die zum Erstellen einer konkreten Syntax genutzt werden können. So spezifiziert zum Beispiel die abstrakte Syntax der UML, dass Klassendiagramme über Klassen- und Interfacebeschreibungen verfügen und welche Elemente in ihnen erlaubt sind.

Aus der abstrakten Syntax kann dann die konkrete Syntax derart abgeleitet werden, dass beispielsweise Interfaces immer durch ein Rechteck mit dem Schlüssel «Interface» dargestellt werden. Eine abstrakte Syntax kann allerdings durch mehrere konkrete Syntaxen implementiert werden. So muss in dem vorhergehenden Beispiel ein Interface...
nicht unbedingt durch eine grafische Repräsentation symbolisiert werden, sondern kann auch als Text, zum Beispiel als XML, dargestellt werden. Abbildung 2 veranschaulicht dieses Beispiel.

Eine abstrakte Syntax kann demzufolge durch verschiedene konkrete Syntaxen, grafisch oder textuell, dargestellt werden [vgl. Gruhn et al. 2006, S.68].

2.4 Meta²-Modell


### Abbildung 3 - Beispiel Stadtplan


### 2.5 MDSD – Model Driven Software Development


„*Modellgetriebene Softwareentwicklung (Model Driven Software Development, MDSD) ist ein Oberbegriff für Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugen.*“

Grundlage der MDSD bilden also formale Modelle [vgl. 2.2], aus denen mittels geeigneter Werkzeuge Quellcode erzeugt wird. Dieser Quellcode muss nicht zwangsläufig direkt zu einem lauffähigen Programm führen. Ziel der MDSD ist es aber
Modelle derart zu spezifizieren, dass manuelle Änderungen am Quellcode nicht mehr benötigt werden. Um aus dem Modell Quellcode zu erzeugen, werden Generatoren benutzt [vgl. 2.5.3.4]. Dieser Vorgang nennt sich Model-to-Code Transformation (M2C) oder auch Model-To-Text Transformation (M2T) [vgl. 2.5.3.4].

Neben der Abkürzung MDSD haben sich auch noch weitere Begriffe für modellgetriebene Softwareentwicklung etabliert. So finden sich ebenfalls die Abkürzungen MDD (Model Driven Development) und MDSE (Model Driven Software Engineering), welche synonym zu modellgetriebener Softwareentwicklung genutzt werden können. Innerhalb dieser Arbeit wird ausschließlich die Abkürzung MDSD verwendet. Die nachfolgenden Kapitel beschäftigen sich mit den grundlegenden Begriffen der MDSD.

### 2.5.1 Domänen und domänenspezifische Sprachen


Domänenspezifische Sprachen (Domain Specific Language, DSL) sind auf eine bestimmte Domäne zugeschnitten. Sie basieren auf der in dem Meta-Modell der Domäne spezifizierten abstrakten Syntax und statischen Semantik und setzen diese durch konkrete Implementierungen um. DSLs können sowohl grafisch er, textueller als auch hybrider Natur sein.

Domänenspezifische Sprachen stehen im Gegensatz zu allgemeinen Sprachen wie der UML. Ihr Ziel ist es, nur den für die Lösung eines Problems notwendigen Umfang zu implementieren. Sie sind deshalb in der Regel nicht auf eine andere Domäne übertragbar. Ein weiterer Vorteil liegt darin, dass sie von Domänenspezialisten leichter erlernbar sind als universelle Sprachen, da sich das Vokabular der Syntax an dem Fachvokabular der Domäne orientiert und die Semantik der DSL versucht die Sachverhalte der Domäne wirklichkeitsgetreu wiederzugeben. Sie können als Programmiersprachen der Domäne angesehen werden, was es ihnen erlaubt wesentlich einfacher Programme zu erstellen, da sie weniger abstrakt und mehr auf das Problem

2.5.2 Plattformen


2.5.3 Modelle der MDSD


2.5.3.1 CIM – Computation Independent Model


2.5.3.2 PIM – Platform Independent Model

späteren konkreten Implementierung. Das PIM ist nicht an die sprachliche Umsetzung gebunden, also ob ein System in Java, C++ oder C implementiert wird.


2.5.3.3 PSM – Platform Specific Model


2.5.3.4 Modell-Transformationen


2.5.4 Generatoren


Neben selbst implementierten Generatoren bietet sich auch die Nutzung von Generator-Frameworks an, welche mit Hilfe von Templates einfach und effizient Modelle transformieren können [vgl. Stahl et al. 2007, S.12 ff.].

Generatoren basieren immer auf einer Plattform, da sie das Quellmodell um die plattformspezifischen Details erweitern. Andernfalls wäre das Quellmodell gleich dem Zielmodell, was den Vorgang der Transformation überflüssig machen würde.

2.5.5 Vor- und Nachteile der MDSD


### 2.5.6 Anverwandte Disziplinen

In den folgenden Kapiteln, sollen zum besseren Verständnis, verwandte Technologien zur modellgetriebenen Softwareentwicklung aufgezeigt und Unterschiede beleuchtet werden.
2.5.6.1 Generative Programmierung


2.5.6.2 MDA – Model Driven Architecture


Wie bei der Syntax und bei der Semantik, gibt die OMG auch bei der Transformation von Modellen die Technologie vor. Hierbei wird *Query View Transformation* (QVT) als Sprache für die Model-to-Model-Transformation genutzt, welche Bestandteil der OMG MOF Spezifikation ist.

Der große Unterschied zwischen MDSD und MDA besteht also in der eingeschränkten Freiheit, die auf Grund der Vorgaben und Spezifikationen seitens der OMG vorhanden ist, bei der Wahl der einzusetzenden Technologien. Andererseits bietet die MDA aber auch einen bereits vordefinierten Workflow und standardisierte Werkzeuge für den modellgetriebenen Softwareprozess.

² http://www.omg.org
³ Mit Hilfe der OCL können Bedingungen (*Constraints*) für das Modell erstellt werden, welche einzelne Bereiche des Modells einschränken oder aber eingrenzen [vgl. Rupp et al. 2006, S. 106 ff.].
2.6 UML – Unified Modeling Language


2.6.1 Diagrammwelten

<table>
<thead>
<tr>
<th>Diagramm</th>
<th>Einsatzbereich</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klassendiagramm</td>
<td>Klassendiagramme beschreiben, aus welchen Klassen das System modelliert werden kann und in welchen Beziehungen die Klassen zueinander stehen.</td>
</tr>
<tr>
<td>Paketdiagramm</td>
<td>Paketdiagramme dienen dem Strukturieren von Projekten, indem sie einzelne Einheiten in Pakete gliedern.</td>
</tr>
<tr>
<td>Objektdiagramm</td>
<td>Objektdiagramme zeigen einen aktuellen Zustand von Objekten in einem System. Sie zeigen die instanzierten Klassen mit ihren aktuellen Parametern und Beziehungen zueinander.</td>
</tr>
<tr>
<td>Kompositionsstruktur-diagramm</td>
<td>Kompositionsstrukturdiagramme stellen das Innenleben von Systemteilen, zum Beispiel Klassen, dar.</td>
</tr>
<tr>
<td>Komponentendiagramm</td>
<td>Komponentendiagramme zeigen die Komponenten des Systems und ihre Schnittstellen.</td>
</tr>
<tr>
<td>Verteilungsdiagramm</td>
<td>Mit Verteilungsdiagrammen wird die Verteilung der einzelnen Softwarekomponenten auf Hardwarekomponenten dargestellt. Zusätzlich werden die Kommunikationsknoten zwischen den einzelnen Strukturelementen sichtbar gemacht.</td>
</tr>
<tr>
<td>Zustandsdiagramm</td>
<td>Gibt einen Überblick über die möglichen Zustände eines Teilsystems, zum Beispiel eines Objektes. Einzelnen Zuständen können Verhalten zugeordnet werden, jeweils, wenn ein Objekt einen Zustand erreicht, ihn verlässt oder sich in ihm befindet.</td>
</tr>
<tr>
<td>Sequenzdiagramm</td>
<td>Sequenzdiagramme stellen zeitlich den Nachrichtenaustausch zwischen Ausprägungen, zum Beispiel zwischen Objekten oder Methoden dar. Es können sowohl synchrone als auch nicht-synchrone zeitliche Abläufe dargestellt werden.</td>
</tr>
<tr>
<td>Kommunikations-diagramm</td>
<td>Kommunikationsdiagramme stellen die Informationen dar, welche von Kommunikationspartnern ausgetauscht werden. Im Gegensatz zum Sequenzdiagramm steht der Gesamtüberblick über die Nachrichten und nicht die zeitliche Abfolge im Vordergrund.</td>
</tr>
</tbody>
</table>

Tabelle 1 - Die 13 Diagramme der UML 2
[vgl. Gruhn et al. 2006]
2.6.2 MOF – Meta Object Facility


Zusätzlich zur Modellebenen-Repräsentation beinhaltet das MOF auch die Spezifikation eines auf XML basierten Austauschformats zur Darstellung von MOF-kompatiblen Datenstrukturen – das XML Metadata Interchange [XMI, vgl. 2.8.4.5].

2.7 OSGi Service Platform

Die OSGi (Open Services Gateway Initiative) Service Plattform ist eine Java-basierte, hardwareunabhängige Softwareplattform für die dynamische Nutzung und Verwaltung von Softwarekomponenten (Bundles) und Diensten (Services). Genauer gesagt handelt es sich bei der OSGi um eine Spezifikation durch die OSGi Alliance, welche seit 1999 entwickelt wird [vgl. Wütherich et al. 2008, S. 12].

Die Spezifikation erlaubt es Bundles und Services zur Laufzeit zu installieren/deinstallieren bzw. zu starten und zu stoppen, also ohne dass die Plattform gestoppt werden muss. Diese Flexibilität hat maßgeblich zum Erfolg von OSGi beigetragen.

Ziel dieser Plattform ist es, die Abhängigkeiten zwischen einzelnen Komponenten eines Softwaresystems zu reduzieren. Die Grundlage der Service Plattform bildet dabei das OSGi-Framework, welches die dynamische Verwaltung von Bundles und Services übernimmt.

2.7.1 Einsatzgebiete

Historisch betrachtet stammt OSGi aus dem Bereich der Gebäudeautomatisierung. Dort diente und dient es, in Form eines Internet Residential Gateways, als Basis für die Vernetzung von einzelnen Systemen untereinander und als Gateway zum Internet. Dadurch können hausinterne Anlagen über das Internet ferngesteuert werden. In seiner Funktion als Residential Gateway stellt OSGi die Schnittstelle von einem externen

Diesen Vorteil haben auch andere Industriezweige erkannt, in denen sich OSGi immer mehr verbreitet. So hat es bereits in den Sektor der mobilen Endgeräte Einzug gehalten, indem es als Basis für Aktualisierungen von Smartphones genutzt wird. Auch im Bereich der Telematik findet eine OSGi-Plattform Anwendung. So verwenden zum Beispiel BMW und Volvo OSGi-Implementierungen für verschiedenen Produkte [vgl. OSGi 2009].


### 2.7.2 OSGi-Framework

Das OSGi-Framework bildet die Grundlage einer OSGi-Implementierung, indem es einen Container für die Verwaltung und Nutzung von Services und Bundles zur Verfügung stellt. Das Framework gliedert sich in mehrere Schichten [vgl. Abbildung 6].


---

Die Modulschicht, als unterste Ebene im OSGi-Framework, stellt die Modularisierung der Plattform zur Verfügung. Integraler Bestandteil dieser Sicht sind die Bundles. Aufgabe der Modulschicht ist es, die Abhängigkeiten zwischen den einzelnen Bundles herzustellen, indem sie die in der Konfiguration des Bundles spezifizierten Importe und Exporte verwaltet und zur Verfügung stellt. Die Modulschicht sorgt also für die Verknüpfung der sonst lose gekoppelten Komponenten.


Um die Ausführungsrechte von Bundles steuern zu können, implementiert ein OSGi-Container eine vertikale Sicherheitsschicht. Über diesen Mechanismus ist es möglich gezielt einzelnen Bundles Rechte zuzuweisen bzw. ihnen diese zu entziehen.

2.7.3 Bundles

muss die Methoden `start(BundleContext)` und `stop(BundleContext)` umsetzen, damit über den `Bundle-Context` eine Verbindung zwischen OSGi-Framework und Bundle hergestellt werden kann.

Die in Bundles enthaltenen Klassen und Funktionalitäten sind nicht automatisch für alle anderen Bundles zugängig. Um die Klassen und Ressourcen für andere zur Verfügung zu stellen, müssen die einzelnen Komponenten exportiert werden. Dadurch kann der Zugriff auf ein Bundle explizit gesteuert werden.

Technisch gesehen handelt es sich bei einem Bundle um nichts anderes als ein normales Java-Archiv (jar), welches um OSGi-spezifische Informationen erweitert wird. Diese Informationen werden innerhalb einer Manifest-Datei (Manifest.MF) deklariert, welche sich innerhalb des META-INF Ordners eines jeden Bundles befindet. In ihr werden alle Bundle-spezifischen Informationen wie zum Beispiel die Version, die Activator-Klasse oder die zur Verfügung gestellten Dienste deklariert.


```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: Extension Plug-in
4  Bundle-SymbolicName: org.mftech.diagram.uml.class.diagram.extension;singleton:=true
5  Bundle-Version: 1.0.0
6  Bundle-Activator: classdiagram.diagram.extension.Activator
7  Require-Bundle: org.mftech.diagram.uml.class.diagram;bundle-version="1.0.0",
8  Import-Package: org.eclipse.emf.transaction,
9                org.eclipse.emf.runtime
10                org.eclipse.gmf.runtime
```

**Listing 1 - OSGi-Manifest**


![Abbildung 8 - Bundle Lebenszyklus](image)

Hierzu wird der in der Manifest-Datei definierte Activator aufgerufen, genauer gesagt dessen Start-Methode ausgeführt. Wird dieser erfolgreich beendet, so ist das Bundle aktiv und kann genutzt werden. Die Activator-Schnittstelle wird ebenfalls genutzt, um ein Bundle wieder zu stoppen, wodurch es automatisch in den Zustand *resolved* zurück versetzt wird.

### 2.7.4 Services


### 2.7.5 OSGi-Implementierungen


Die unten aufgeführte Tabelle gibt einen groben Überblick über derzeitige Open-Source OSGi-Implementierungen.

<table>
<thead>
<tr>
<th>Name</th>
<th>Hersteller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Felix</td>
<td>Apache Software Foundation</td>
</tr>
<tr>
<td>Equinox</td>
<td>Eclipse Foundation</td>
</tr>
<tr>
<td>Knopflerfish</td>
<td>MakeWave&lt;sup&gt;7&lt;/sup&gt;</td>
</tr>
<tr>
<td>Prosyst mBedded Server Equinox Edition</td>
<td>Prosyst&lt;sup&gt;8&lt;/sup&gt;</td>
</tr>
<tr>
<td>Oscar OSGi</td>
<td>OW2 Consortium&lt;sup&gt;9&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

Tabelle 2 - Open-Source OSGi-Implementierungen

Da OSGi grundlegend eine komponentenbasierte Architektur nutzt, werden Abhängigkeiten zwischen einzelnen Systemteilen auf ein Minimum reduziert.

<sup>7</sup> http://www.makewave.com/site.en/
<sup>8</sup> http://www.prosyst.com/products/osgi_se_equi_ed.html
<sup>9</sup> http://forge.ow2.org/projects/oscar/

Weil Bundles auch in verschiedenen Versionen ausgeliefert werden können, ist es möglich auch Systemaktualisierungen effektiv durchzuführen, was eine hohe Wartbarkeit, Skalierbarkeit und Flexibilität des auf OSGi-basierten Systems erlaubt.


### 2.7.6 OSGi und SOA

In der Fachwelt tritt zuweilen die Behauptung auf, OSGi sei die Umsetzung einer serviceorientierten Architektur (SOA) innerhalb der Virtuellen Maschine. Oftmals wird diese These allerdings stark diskutiert [vgl. Voelter Blog 2007].

Der Begriff einer serviceorientierten Architektur hat sich leider nicht in einer einheitlichen Definition durchgesetzt. Im Allgemeinen wird unter einer SOA aber eine auf Diensten basierende Architektur verstanden, in der die Dienste ihre Funktionen durch definierte Schnittstellen anderen Diensten zur Verfügung stellen. [Melzer et al. 2008] liefert hierzu folgende Definition:

„Serviceorientierte Architekturen, kurz SOA, sind das abstrakte Konzept einer Software-Architektur, in deren Zentrum das Anbieten, Suchen und Nutzen von Diensten über ein Netzwerk steht.“

Anwendungen in einer SOA werden durch die Orchestrierung von Diensten zusammengebaut und Dienste können ebenfalls aus solchen bestehen. Ein wichtiger Aspekt einer SOA ist die lose Bindung zwischen den einzelnen Diensten, da die Kommunikation ausschließlich über die Schnittstellen erfolgt. Hierbei können in der Praxis verschiedene Technologien (Webservices, CORBA, REST etc.) genutzt werden,
wobei sich zur Zeit am stärksten Websevices für die Implementierung einer SOA durchgesetzt haben.


“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties” [Szyperski et al. 2002, S.41].


Alle diese Eigenschaften vereint auch ein Service im Kontext von SOA auf sich. Ebenso wie bei der Software-Komponente, spielt die konkrete Implementierung eines Dienstes keine Rolle, solange die Vorgaben der Schnittstelle erfüllt sind. Daneben werden Services auch als wiederverwendbare Konstrukte genutzt, um eine gesamte Anwendung zu erstellen, deren einzelne Elemente wiederverwendbar und austauschbar sind. Im Gegensatz zur Komponente, ist ein Service aber nicht kontextabhängig. Software-Komponenten sind programmiersprachenabhängig und aus diesem Grund wesentlich stärker technologieorientiert. Services hingegen nutzen ein interoperables Austauschformat, wie zum Beispiel XML, weshalb die Programmiersprache für die Implementierung der Service-Schnittstelle nicht von Bedeutung ist. Diese Aussage wird durch folgende Definition gestützt:


OSGi, basierend auf dem Komponentenmodell, bietet mit den Bundles und Services ebenfalls schnittstellenorientierte Softwareteile, also Komponenten ähnlich einer SOA an. Durch die explizite Bindung an Java können die einzelnen Komponenten von OSGi
allerdings nicht losgelöst von der Virtuellen Maschine genutzt werden. Weder ein Bundle noch ein OSGi-Service können also als Dienst bezüglich einer SOA betrachtet werden. Beide Begriffe und Definition können zwar als stark verwandt, aber nicht als identisch angesehen werden. Somit stellt OSGi auch keine Implementierung einer serviceorientierten Architektur dar.

### 2.8 Eclipse


\(^{10}\) [http://www.eclipse.org/cdt/](http://www.eclipse.org/cdt/)

\(^{11}\) Projekte im Incubator werden zu vollständigen Open-Source Projekten entwickelt.
<table>
<thead>
<tr>
<th>Name</th>
<th>Anwendungsgebiet</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIRT</td>
<td>Business Intelligence and Reporting Tools; Erstellung von Reports; <a href="http://www.eclipse.org/birt">http://www.eclipse.org/birt</a></td>
</tr>
<tr>
<td>Buckminster</td>
<td>Framework zum automatischen Bauen und Deployen von Projekten; <a href="http://www.eclipse.org/buckminster">http://www.eclipse.org/buckminster</a></td>
</tr>
<tr>
<td>CDT</td>
<td>C/C++ Development Tooling; IDE für C/C++; <a href="http://www.eclipse.org/cdt">http://www.eclipse.org/cdt</a></td>
</tr>
<tr>
<td>DLTK</td>
<td>Dynamic Languages Toolkit; Toolkit für dynamische Sprachen wie Tcl, Ruby oder Python; <a href="http://www.eclipse.org/dltk">http://www.eclipse.org/dltk</a></td>
</tr>
<tr>
<td>DSDP TM</td>
<td>Device Software Development Platform – Target Management; Remote Verwaltung von Objekten; <a href="http://www.eclipse.org/dsdp/tm/">http://www.eclipse.org/dsdp/tm/</a></td>
</tr>
<tr>
<td>DTP</td>
<td>Data Tools Platform; <a href="http://www.eclipse.org/datatools/">http://www.eclipse.org/datatools/</a></td>
</tr>
<tr>
<td>ECF</td>
<td>Eclipse Communication Framework; <a href="http://www.eclipse.org/ecf/">http://www.eclipse.org/ecf/</a></td>
</tr>
<tr>
<td>Eclipse Project</td>
<td>Hauptprojekt von Eclipse; <a href="http://www.eclipse.org/eclipse/">http://www.eclipse.org/eclipse/</a></td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modeling Framework; Basis für modellgetriebene Softwareentwicklung; <a href="http://www.eclipse.org/emf/">http://www.eclipse.org/emf/</a></td>
</tr>
<tr>
<td>EMFT</td>
<td>Eclipse Modeling Framework Technologies; bietet Tools zum Verarbeiten von Modellen; <a href="http://www.eclipse.org/modeling/emft">http://www.eclipse.org/modeling/emft</a></td>
</tr>
<tr>
<td>EPP</td>
<td>Eclipse Package Projekt; <a href="http://www.eclipse.org/epp">http://www.eclipse.org/epp</a></td>
</tr>
<tr>
<td>GEF</td>
<td>Graphical Editor Framework; Entwicklung von grafischen Editoren; <a href="http://www.eclipse.org/gef/">http://www.eclipse.org/gef/</a></td>
</tr>
<tr>
<td>GMF</td>
<td>Graphical Modeling Framework; Entwicklung von grafischen Editoren basieren auf EMF; <a href="http://www.eclipse.org/gmf/">http://www.eclipse.org/gmf/</a></td>
</tr>
<tr>
<td>MDT</td>
<td>Model Development Tools; u.a. Support für BPMN, UML oder OCL; <a href="http://www.eclipse.org/modeling/mdt/">http://www.eclipse.org/modeling/mdt/</a></td>
</tr>
<tr>
<td>M2M</td>
<td>Unterstützt Model-to-Model Transformationen; <a href="http://www.eclipse.org/m2m/">http://www.eclipse.org/m2m/</a></td>
</tr>
<tr>
<td>M2T</td>
<td>unterstützt Model-to-Text Transformationen; <a href="http://www.eclipse.org/modeling/m2t/">http://www.eclipse.org/modeling/m2t/</a></td>
</tr>
<tr>
<td>MyLyn</td>
<td>Zur Verwaltung von Aufgaben in einem Software Projekt; <a href="http://www.eclipse.org/mylyn/">http://www.eclipse.org/mylyn/</a></td>
</tr>
<tr>
<td>STP</td>
<td>SOA Tools Platform Projekt; unterstützt SOA-orientiertes Entwickeln; <a href="http://www.eclipse.org/stp/">http://www.eclipse.org/stp/</a></td>
</tr>
<tr>
<td>Subversive</td>
<td>Versionsmanagement auf Basis von Subversion; <a href="http://www.eclipse.org/subversive/">http://www.eclipse.org/subversive/</a></td>
</tr>
<tr>
<td>TPTP</td>
<td>Testumgebung für Eclipse; <a href="http://www.eclipse.org/tptp/">http://www.eclipse.org/tptp/</a></td>
</tr>
<tr>
<td>WTP</td>
<td>Web Tools Plattform; Unterstützung bei der Entwicklung von webbasierten Anwendungen; <a href="http://www.eclipse.org/webtools/">http://www.eclipse.org/webtools/</a></td>
</tr>
</tbody>
</table>

Tabelle 3 - Überblick über die Eclipse Projekte im Ganymede Release
[Quelle: vgl. Teufel 2008, S.14]
2.8.1 Die Plattform – Equinox


Zwar basiert Equinox auf Eclipse, kann aber komplett losgelöst von diesem als eigenständige OSGi-Implementierung genutzt werden. Dabei ist es mit einer Größe von 4,7 Megabytes (Release 3.4\(^{12}\)) sehr schlank gehalten.


Für die Verwaltung des Containers setzt Equinox eine einfache Konsole als Management Agent ein, welche mit wenigen Befehlen auskommt [vgl. Abbildung 7].

Wie aus Abbildung 9 ersichtlich, basiert jede Entwicklung von Eclipse auf Equinox als unterste Schicht im Eclipse-Architektur-Modell. Das bedeutet, dass jede auf dieser Plattform basierende Software auch ihre Vorteile, zum Beispiel die des Plugin-Konzepts, nutzen kann.

\(^{12}\) http://download.eclipse.org/eclipse/equinox/
2.8.2 RCP – Rich Client Platform


2.8.3 PDE – Plugin Development Environment

Eines der Projekte, welches auf der Eclipse Plattform Equinox aufbaut, ist die Plugin Entwicklungsumgebung (*Plugin Development Environment, PDE*). PDE ist eins von fünf Sub-Projekten des Eclipse-Projektes und stellt nützliche Tools und User-Interfaces für die Bearbeitung von Plugins zur Verfügung und ist somit essentieller Bestandteil einer jeden Eclipse Entwicklung. Da Plugins nahezu identisch mit OSGi-Bundles sind [vgl. 2.8.1 und 2.8.3.1], kann PDE auch außerhalb von Eclipse für die Entwicklung von OSGi-Anwendungen genutzt werden.
Die Plugin Entwicklungsumgebung unterteilt sich in drei Teile – PDE Build, PDE API Tools und PDE UI. Die ersten beiden Projekte unterstützen den Entwickler beim Build-Prozess und bieten hilfreiche Funktionen, um beispielsweise die Versionen der Plugins zu verwalten. Das PDE UI wiederum bietet eine Schaltzentrale für die komplette Verwaltung der Plugin-Entwicklung [vgl. Abbildung 10].

![Abbildung 10 - PDE User Interface](image)

Hierbei stellt es die Informationen aus den unterschiedlichen Konfigurationsdateien eines Plugins aufbereitet dar und erlaubt auf einfache Weise deren Manipulation. Dazu bietet es verschiedene Editoren an, mit denen auch direkt die Dateien im Text-Format bearbeitet werden können. Im Folgenden sollen die für die Arbeit notwendigen Begriffe im Rahmen der Plugin-Entwicklung beschrieben und erläutert werden.

### 2.8.3.1 Plugins


Plugins können, wie Bundles, eine Activator-Klasse besitzen, welche die Methoden zur Steuerung des Lebenszyklus des Plugins bereitstellt. In der Regel erbt diese von der
abstrakten Basisklasse Plugin, welche das Interface org.osgi.framework.BundleActivator implementiert und somit eine Implementierung der OSGi-Spezifikation ist.


2.8.3.2 Extensions


In diesem Zusammenhang lässt sich die Analogie zu einer Steckdose und einem Stecker verwenden. Der Editor entspricht dem Stecker (Extension), welcher mit einer passenden Steckdose (Extension Point) verbunden werden kann [vgl. Abbildung 11].

Im Ergebnis kann ein neuer Editor in das System eingebaut werden, welcher alle Eigenschaften von bestehenden Editoren nutzt. So ist es in Eclipse beispielsweise möglich bestimmte Dateiendungen mit einem Editor so zu verbinden, dass sich bei Auswahl der

Abbildung 11 - Plugin Mechanismus
Grundlagen

Datei der richtige Editor öffnet. Diese und andere Informationen können deklarativer in der Extension beschrieben werden, sodass dieses Verhalten nicht eigenständig programmiert werden muss.

Dabei ist zu beachten, dass Erweiterungen nicht allein auf UI-Elemente beschränkt sind und in jeglicher Form innerhalb einer Software genutzt werden können.

2.8.3.3 Extension Points


Abbildung 12 - Nutzungs- und Abhängigkeitsrichtung bei Vererbung

Soll nun aber a die aktive Komponente sein, die über zusätzliche Informationen von b erweitert wird, so ist dies mit dem Vererungsmodell nicht zu realisieren. Klasse B kann
alle Informationen der Basisklasse nutzen. Eine umgekehrte Nutzung ist allerdings nicht möglich. Ebenfalls entsteht durch das Vererbungskonzept eine starke Abhängigkeit zwischen den beiden Komponenten. Diese Abhängigkeit kann zwar durch die Nutzung von Interfaces reduziert werden, bestehen bleibt allerdings das Problem, dass Plugin a nicht als eigenständige Komponente dynamisch durch die Informationen ergänzt werden kann.

Dieses Problem kann durch die Nutzung von Extension Points gelöst werden. Hierbei kann a als eigenständige Komponente laufen, zum Beispiel als lauffähiger Editor. Bei Bedarf kann, sogar zur Laufzeit, das Verhalten von a geändert werden, indem es die Methoden der Klasse B benutzt. Dabei dreht sich also die Nutzungsrichtung um [vgl. Abbildung 13].

![Abbildung 13 - Nutzungs- und Abhängigkeitsrichtung bei Extension Points](image)

Ermöglicht wird dies, indem Plugin a die Schnittstellen definiert, an denen es sich erweitern lässt. Diese Schnittstellen (Extension Point) wird deklarativ in der plugin.xml beschrieben. Sie verweist auf ihre Definition, welche in Form einer XML-Schema Beschreibung erstellt und im Plugin hinterlegt wird. Plugin b kann diese benutzen, um sich am Plugin a anzumelden [vgl. Hennig et al 2008, S. 19 ff.]. Zur Laufzeit kann Plugin a eine globale Registrierung (Extension-Registry) abfragen, ob sich Erweiterungen angemeldet haben. Die Extension-Registry wird innerhalb von Equinox als OSGi-Service zur Verfügung gestellt. Da die Beschreibung von Extension Point und Extension rein deklarativ erfolgen und die genutzten Methoden anhand von Schnittstellendefinitionen vereinbart werden, sind die Komponenten a und b vollständig von einander entkoppelt, was dem der OSGi zugrunde liegenden Komponentenmodell entspricht.
2.8.3.4 Fragmente


2.8.3.5 Ansichten, Editoren und Perspektiven

In jeder RCP (Rich Client Platform), also auch in der Eclipse IDE, spielen drei Konzepte eine wichtige Rolle – Ansichten, Editoren und Perspektiven. Ansichten (Views) und Editoren sind Bestandteile einer jeden Workbench-Seite, also dem Fenster der RCP. Editoren und Ansichten ähneln sich stark. Beide können genutzt werden, um Daten zu manipulieren. Dabei kommen aber Editoren verstärkt zum Einsatz wenn Daten direkt editiert werden sollen, beispielsweise um Texte, Quellcodes, oder grafische Modelle zu bearbeiten. Views hingegen dienen meist vorrangig zur Anzeige von Daten, zum Beispiel um die Paket-Struktur eines Projektes darzustellen. Sie können aber auch Daten manipulieren. Perspektiven bieten eine vordefinierte Anordnung von Editoren und Views, die unter einem bestimmten Namen hinterlegt ist. Wird in eine Perspektive gewechselt, so werden genau die Views und Editoren angezeigt, die typisch für diese Perspektive sind [vgl. Sippel 2008; S.172 ff.].
2.8.4 EMF – Eclipse Modeling Framework


Dabei unterteilt sich EMF in drei Bestandteile - dem EMF.Core, dem EMF.Edit und dem EMF.Codegen.

2.8.4.1 EMF.Core – Ecore


Grundlegend kann das Ecore auf vier Elemente reduziert werden, EClass, EAttribute, EReference und EDataType, deren Zusammenhang in Abbildung 14 dargestellt ist. Diese Namen weisen nicht grundlos eine starke Ähnlichkeit zu den Bezeichnern der UML-Klassendiagramm-Spezifikation auf. Das EMF-Core ist eine vereinfachte Form des Klassendiagramm Modells, welches auf dem MOF-Standard basiert. Um genau zu sein, kann Ecore als Implementierung von EMOF betrachtet werden [vgl. 2.6.2].

![Abbildung 14 – vereinfachtes EMF Ecore Modell](Quelle: vgl. Merks et al. 2009)

EClasses repräsentieren die Modell-Klassen. Sie beinhalten Attribute (EAttribute) und Referenzen (EReference). EAttributes sind vergleichbar mit den Attributen aus einem Klassendiagramm. Sie bestehen aus einem Namen und einem Typ (EDataType). Im
Grundlagen

Gegensatz dazu stellen EReferences eine Assoziation zwischen Klassen dar. EDataTypes werden genutzt, um den Datentyp eines EAttributes darzustellen. Hierbei kann der EDataType sowohl ein primitiver Datentyp (int, double, etc), eine beliebige Java-Klasse, als auch eine in dem Modell definierte Klasse sein.


### 2.8.4.2 Generator Modell


### 2.8.4.3 EMF.Edit und EMF.Codegen


2.8.4.4 Resource und ResourceSet


2.8.4.5 EMF und XMI


2.8.4.6 Dynamic EMF


```java
for (EAttribute attribute: oldObject.eClass().getEAllAttributes())
{
    newObject.eSet(attribute, oldObject.eGet(attribute));
}
```

Listing 2 - Dynamic EMF

### 2.8.5 Exkurs – SWT, JFace und Draw2D

Da die nachfolgenden Eclipse-Projekte durch die Nutzung grafischer Frameworks geprägt sind, soll an dieser Stelle ein Überblick über die bestehenden grafischen Bibliotheken geschaffen werden.

In der Java-Welt gibt es verschiedene Bibliotheken, welche es erlauben, grafische User-Interfaces zu erstellen. Das *Abstract Window Toolkit (AWT)* war dabei die erste Implementierung. AWT als schwergewichtiges Toolkit bezeichnet, was bedeutet, dass es zur Darstellung der grafischen Elemente die Funktionen benutzt, die direkt vom Betriebssystem angeboten werden. Da Java allerdings plattformunabhängig ist, konnte die AWT Implementierung nur aus dem kleinsten gemeinsamen Nenner der GUI-Elemente aller unterstützten Betriebssysteme bestehen. Auch führte dieser Ansatz dazu, dass die Oberfläche auf unterschiedlichen Systemen anders aussah. Diese Nachteile und weitere Einschränkungen von AWT führten dazu, dass mit Java 1.2 eine neue grafische Bibliothek in den Standard integriert wurde – *Swing*. In Swing wurden alle Komponenten selbst implementiert. Es verfolgt also einen leichtgewichtigen Ansatz, was die GUI unabhängig von den vom Betriebssystem bereitgestellten Elementen macht. Dies hat allerdings den Nachteil, dass das Zeichnen von Elementen sehr ressourcenlastig ist, wodurch auf Swing basierende Anwendungen oft als sehr „schwerfällig“ empfunden werden. Nicht zuletzt diese Tatsache ist mit dafür
grundlagen 43


Da das Erstellen von komplexen GUIs mit sämtlichen Ereignisbehandlungen unter Umständen recht aufwendig werden kann und komplexere Widgets (komplexe Dialoge und Wizards) immer wieder neu erstellt werden mussten, wurde eine zusätzliche Bibliothek geschaffen, die Teile von SWT kapselt und einfachere Implementierungen erzeugen kann. JFace bildet also eine Implementierungsschicht, welche direkt auf SWT aufsetzt. Es vereinfacht die Entwicklung von grafischen Oberflächen wesentlich und erlaubt es übersichtlicheren Code zu erstellen [vgl. Sippel 2008, S.81 ff.].


2.8.6 GEF – Graphical Editing Framework


Grundlage für die Umsetzung des MVC Pattern bilden in GEF die EditParts, welche die Funktion von Controllern übernehmen. EditParts sind Klassen, welche die grafische Repräsentation eines Objektes erzeugen. Sie stellen also die Definition einer Grafik mit
Hilfe von Draw2D dar. Diese Darstellungen erben immer von der Basisklasse `Figure` [vgl. Mammana 2008].

Neben der Erzeugung der View stellen die EditParts die Verbindung zum Modell her. In GEF kann das Model frei definiert werden und liegt in Form von einfachen Java-Beans vor.

Abbildung 15 - Darstellung von MVC in GEF

GEF ist sehr fein granular. Das bedeutet, dass für die Verwaltung einzelner Objekte jeweils ein eigener EditPart und eine eigene Figur existieren. Besteht zum Beispiel das Datenmodell `Klassen` aus Attributen und Operationen, so können für diese eigene Figuren und EditParts implementiert werden.


### 2.8.6.1 Requests, Commands und EditPolicies


Abbildung 16 - Ausführen eines Kommandos

2.8.7 GMF – Graphical Modeling Framework

Das *Graphical Modeling Framework* (GMF) bildet die Symbiose zwischen EMF und GEF. Hierbei erlaubt es anhand eines einfachen vordefinierten Workflows grafische Editoren zu erstellen, welche auf einem EMF-Modell aufbauen. EMF liefert also die Bearbeitung des Daten-Modells, während GEF Methoden zur Behandlung der grafischen Repräsentation besteuert. Dabei können sehr schnell und flexibel lauffähige Editoren erstellt werden. In den folgenden Kapiteln werden die für die Arbeit notwendigen Begrifflichkeiten im Umfeld von GMF erläutert.
2.8.7.1 GMF-Tooling


Abbildung 17 - Einfacher mit GMF erzeugter Editor

Ein weiteres Modell, das GMF-Tooling-Model, definiert den Aufbau der Werkzeugleiste [vgl. Abbildung 17, Nr. 2]. Auch hier kann anhand einer baumbasierten Struktur das Aussehen und Verhalten des Editors konfiguriert werden. Mit diesem

Als Eingabe für den GMF-Generator, der aus der Spezifikation den Editor erzeugt, dient ein weiteres Model – das GMF-GenModel. Es wird aus dem GMF-Mapping-Modell erzeugt.

Abbildung 18 - GMF-Modelle

In Abbildung 18 wird der Zusammenhang zwischen EMF und GMF noch einmal grafisch dargestellt. GMF setzt dabei auf ein bereits vorhandenes EMF Modell auf. Während EMF für die Generierung des Model-Codes und des Edit-Codes zuständig ist [vgl. 2.8.4.2], erzeugt GMF den Quellcode für den Editor und baut dabei auf die bereits erstellen Plugins auf. Im Ergebnis entsteht ein Editor basierend auf dem bestehenden Domänenmodell, der im weiteren Schritt um weitere Features ergänzt werden kann.
2.8.7.2 GMF-Runtime


![Abbildung 19 - GMF-Architektur](Quelle: vgl. GMF 2009b)

### 2.8.7.3 Figure Deskriptoren

GMF nutzt *Figure Deskriptoren*, um innerhalb des GMF-Graph-Modells das Aussehen der einzelnen Diagrammelemente zu definieren. Für jedes Element wird ein Figuren Deskriptor erstellt. Diese werden in einer *FigureGallery* zusammengefasst.


Um Referenzen zwischen Knoten, also Kanten darzustellen, werden in GMF PolyLine Connections genutzt. PolyLine Connections können Referenzen auf *Decoration Figure* Deskriptoren enthalten, welche ebenfalls in der FigureGallery definiert sind. GMF unterscheidet dabei zwei verschiedene Arten von Dekorationen *PolyLine Decorations* und *Polygon Decorations*. Erstere dienen ausschließlich der Darstellung von linienorientierten Dekorationen. Mit ihrer Hilfe können einfache Pfeile oder Objekte ohne Hintergrund erstellt werden. *Polygon Decorations* dagegen sind immer ausgefüllte Objekte. Das bedeutet, dass der Bereich, der von den Punkten einer Polygonbeschreibung umspannt wird, mit der in der Dekoration angegebenen Farbe ausgefüllt wird. Sie können zum Beispiel genutzt werden, um Darstellungen wie ausgefüllte Rauten zu erstellen. Um das Aussehen der Dekoratoren zu verändern, können Template Points benutzt werden. Ein Template Point definiert einen Punkt im zweidimensionalen Raum, relativ zum Ende der PolyLine. Über die Angabe mehrerer Template Points kann so die Beschreibung komplexer Objekte hinterlegt werden [vgl. GMF 2009a].

2.8.7.4 ElementTypes

Jede Figur innerhalb der FigureGallery kann mehrfach genutzt werden. So enthalten beispielsweise in einem Klassendiagramm sowohl die Klassen als auch die Schnittstellen eine Figur zum Anzeigen von Operationen. Diese wird aber nicht mehrfach definiert, sondern es gibt nur eine Figure, welche innerhalb des GMF-Mapping-Modells entweder dem Klassen- oder Schnittstellen-Modell zugeordnet wird. Dadurch wird die Darstellung (GMF-Graph) vom Datenmodell (semantic und notational Modell) abstrahiert. Erst auf der Ebene des Mappings wird eine Zuordnung getroffen.


2.8.7.5 Vor- und Nachteile von GMF

Das GMF-Framework erlaubt es schnell und effizient Rohformen von grafischen Editoren zu erzeugen, welche für viele Anwendungszwecke bereits vollkommen ausreichend sind. Bei Bedarf kann aufgrund des generischen Rahmenkonzeptes von GMF der Editor beliebig erweitert werden. Im Vergleich zu GEF ist es aber an ein festes Modell gekoppelt. Soll also ein Domänenmodell genutzt werden, welches nicht auf EMF beruht, muss es nach EMF konvertiert oder GEF als Framework für die Erstellung des grafischen Editors genutzt werden.

2.9 Erweiterte Grundlagen

Nach der Einführung in die wichtigsten Konzepte der modellgetriebenen Softwareentwicklung und der Welt von Eclipse werden im folgenden weitere Begriffe und Frameworks erläutert, welche für die umzusetzenden Ziele der Arbeit benötigt werden.
2.9.1 Web Services


13 World Wide Web Consortium; http://www.w3.org/
14 Remote Procedure Call, Aufrufen einer Funktion über Systemgrenzen hinweg. Es ist dabei für den Aufrufer transparent, wo die Ausführung stattfindet und kann auch auf einem anderen Rechner oder System erfolgen [vgl. Tanenbaum 2003, S526 ff.]
2.9.2 Apache Axis


2.9.3 Hibernate


2.9.4 Ajax


\textsuperscript{15} https://www.hibernate.org/
(DOM\(^{16}\)) ermöglicht Ajax es so dynamisch Teile einer Webseite neu zu laden oder auszutauschen.

### 2.10 Zusammenfassung


\(^{16}\) DOM – Technologie, welche es ermöglicht ein HTML- oder XML-Dokument in Form eines Baumes darstellbar und zugreifbar zu machen [vgl. Wenz 2008,S.249 ff].
3 Anforderungsanalyse und -definition

Innerhalb der Arbeit soll eine generische, modellgetriebene Netzwerkkomponente entwickelt werden, mit der es möglich ist GMF-basierte Editoren über ein Netzwerk so zu verbinden, dass den Anwendern ein kollaboratives Arbeiten am grafischen Modellbestand ermöglicht wird. Hierbei sind unterschiedlichste Aspekte, sowohl in Hinsicht auf softwaretechnische Probleme, als auch in Bezug auf die Nutzerfreundlichkeit, bei der Verwendung der Schnittstelle und der modellgetriebenen Erzeugung der dafür notwendigen Erweiterungen, relevant.

Im Folgenden sollen die Anforderungen an das Projekt analysiert und infolgedessen spezifiziert werden. Dabei werden mögliche Technologien betrachtet und auf ihre Verwendbarkeit innerhalb des Projektes hin untersucht.

In Vorbereitung auf die Entwicklung des Konzeptes werden in diesem Kapitel die funktionalen, systemtechnischen Anforderungen an das System erstellt und festgehalten. Diese Spezifikation resultiert in dem in Anhang A festgehaltenen Anforderungskatalog.

Eingangs werden verschiedene Ansätze diskutiert und bezüglich ihrer Nützlichkeit und Umsetzbarkeit im Rahmen des Projektes analysiert. Dies geht einher mit der Definition der für diese Arbeit notwendigen Systemfestlegungen.

3.1 Betrachtung Eclipse-basierter Netzwerk-Frameworks

Das Eclipse Projekt stellt zwei Frameworks zur Verfügung, um netzwerkbasiert Informationen austauschen zu können. In den folgenden Kapiteln sollen beide betrachtet und für eine mögliche Nutzung im Projekt evaluiert werden.

3.1.1 CDO – Connected Distributed Objects

Das Eclipse Project bietet mit Connected Distributed Objects (CDO\textsuperscript{17}) ein Framework, um EMF-basierte Diagramme kollaborativ über ein gemeinsames Repository nutzen zu können. Es ist ein Sub-Projekt des EMFT-Projektes (Eclipse Modeling Framework Technology). Ziel von CDO ist es, das kollaborative Arbeiten an baumbasierten EMF-Projekten zu unterstützen und zu vereinfachen. Dabei setzt CDO für die

\footnotesize{\textsuperscript{17}http://wiki.eclipse.org/CDO}


\textsuperscript{18} http://wiki.eclipse.org/Net4j
Dieses Vorgehen hat zum einen den Vorteil, dass das System eine hohe Performance gewährleisten kann, da die Kommunikation auf ein Minimum reduziert wird. Zum anderen wäre bei sehr großen Modellen eine komplette Übertragung aller Daten mit den heutigen technischen Möglichkeiten undenkbar. So ist CDO für Modellbestände in Größenordnungen von mehreren Gigabytes ausgelegt, was bei einer initialen Übertragung mehrere Stunden in Anspruch nehmen würde. Leider verhindert dieser Mechanismus aber auch, dass kleinere Projekte unabhängig vom Server arbeiten können. Bricht die Verbindung zum Server ab, kann der Client nicht mehr an seinem Modell arbeiten.


3.1.2 ECF – Eclipse Communication Framework


3.2 Generator Frameworks

Das System wird sich in den generativen Entwicklungsprozess des GMF-Frameworks einreihen, dass bedeutet, dass sämtliche Erweiterungen und Quellcodes ebenfalls über
zugrunde liegende Modelle in ausführbaren Programmcode überführt werden. Im Bereich der modellgetriebenen Softwareentwicklung gibt es verschiedene Frameworks, welche Transformationen sowohl von Model zu Modell, als auch von Modell zu Code unterstützten. Im Folgenden werden verschiedene dieser Frameworks zur Generierung von Text aus Modellen (M2T) verglichen und eine Entscheidung für das am geeignetsten in Bezug auf diese Arbeit getroffen.

3.2.1 JET – Java Emitter Templates


3.2.2 XPand – oAW


\(^{19}\) XPath – Abfragesprache für XML-Dokument; entwickelt vom W3C

3.3 Web Frameworks


---

20 http://struts.apache.org/
21 http://java.sun.com/javaee/javaserverfaces/
22 http://shale.apache.org/
23 http://myfaces.apache.org/
24 http://tapestry.apache.org/
25 http://www.springsource.org/
26 http://www.stripesframework.org
die Suchergebnisse von Stellenbörsen miteinander verglichen wurden. Zusätzlich wurde die angebotene Fachliteratur mengenmäßig verglichen.

Für die Analyse der Marktverbreitung sind die Online-Portale von Monster.de27, JobScout2428 und Stepstone29 nach Jobangeboten durchsucht worden, welche Kenntnisse in den einzelnen Frameworks voraussetzen. Dabei zeigte sich ein starker Trend zu Struts und JSF. Die anderen Frameworks werden wesentlich weniger stark von der Industrie nachgefragt. In Abbildung 20 sind die Ergebnisse in einer Übersicht dargestellt

<table>
<thead>
<tr>
<th>Vergleich Job-Börsen (April 2009)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monster.de</td>
</tr>
<tr>
<td>119</td>
</tr>
<tr>
<td>90</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Abbildung 20 - Webframeworks - Vergleich Jobbörsen

Um zusätzlich die Nachfrage zu untersuchen und die angebotenen Kompetenzen der Fachkräfte beurteilen zu können, wurde auch auf der Kontakt-Plattform XING30 nach Angeboten und Nachfragen gesucht. Da die Untersuchungen nicht mit dem Premium-Account von XING ausgeführt wurden, waren die maximalen Suchergebnisse auf 200 beschränkt. Dadurch lässt sich kein direkter Vergleich zwischen JSF und Struts herstellen. Es zeigt sich aber, dass diese beiden Frameworks trotzdem am stärksten nachgefragt werden und auch zu den Kompetenzen der meisten Personen gehören.

27 www.monster.de
28 www.jobscout24.de
29 www.stepstone.de
30 www.xing.de


³¹ www.amazon.de
auch Neuerscheinungen in Englisch mit berücksichtigt werden, wurde die Suche sowohl für deutsch- als auch englischsprachige Werke ausgeführt.

Die eingesetzten Methoden erheben keinen Anspruch auf Vollständigkeit oder können eine genaue Aussage zu einem absoluten Favoriten treffen. Dazu wäre eine wesentlich intensivere Analyse erforderlich, als es im Rahmen dieser Arbeit möglich gewesen wäre. Sie zeigt aber eine gute Tendenz über die Verbreitung der einzelnen Produkte. Demnach finden Struts und JSF die meiste Marktverbreitung.


3.4 Anwendungsszenarien


Dies bietet sich besonders für Firmen an, deren Hauptinnahmequelle im Consulting besteht und deren Experten dementsprechend häufig auf Geschäftsreisen sind. Selbst ohne installiertes Eclipse ist es möglich sich an der Projektarbeit zu beteiligen. Durch das zur Verfügung gestellte Webinterface können dabei sogar Geräte eingesetzt werden, welche nicht für den Einsatz von Eclipse geeignet sind. So ist die Betrachtung des grafischen Modells auch auf mobilen Geräten möglich, wenn diese über einen
geeigneten Netzwerkzugang verfügen. Infolgedessen ist die Betrachtung des Modellbestandes unabhängig von Ort und Zeit.

Neben der kollaborativen grafischen Entwicklung ist aber noch eine Vielzahl weiterer Anwendungsbereiche möglich. Im Managementbereich können in Echtzeit Geschäftsprozesse, Workflows oder Organigramme modelliert werden. Dabei ist es sogar denkbar, dass direkt in den Ablauf des Workflows über einen entfernten Rechner eingegriffen werden kann. Prinzipiell sind also die Anwendungsmöglichkeiten so vielfältig wie die der generierten Editoren.

Abbildung 23 - Anwendungsfall: Überblick


3.5 System Architektur


Abbildung 24 zeigt eine vereinfachte Systemdarstellung der zu entwickelnden Komponenten (gelb dargestellt).
3.6 Systemanforderungen


3.6.1 Kommunikation und Firewall-Transparenz


---

\(^{32}\) Polling – zyklisches Abfragen eines Systemzustandes
3.6.2 Ausfallresistenz, Mobilität und Netzwerkunabhängigkeit


3.6.3 Web-Viewer und Systemverwaltung

Wie in Kapitel 1.2 beschrieben, soll das System über eine Web-Oberfläche den Modellbestand auch unabhängig von Eclipse darstellen können. Hierzu muss eine Möglichkeit geschaffen werden, um die grafischen Elemente des GMF-Editors als Web-Element darstellen zu können. Dies ist besonders schwierig, da das Aussehen des GMF-Editors von den Endanwendern bestimmt wird, also im Vorhinein nicht bekannt ist. Es müssen also über ein generisches, adaptives Verfahren die spezifizierten View-Elemente umgewandelt werden. Da das System sowohl den Server als auch den Netzwerkadapter mittels modellgetriebener Techniken erzeugt [vgl. 3.6.6], bietet es sich an, auch die Web-Elemente in diesem Vorgang zu generieren. Dabei kann auf das gmfgraph-Modell zugegriffen werden, welches alle Informationen für das Aussehen der


### 3.6.4 Synchronisation und Ressourcenverwaltung


3.6.5 User-Management und Sicherheit


3.6.6 Generierung der Komponenten


3.6.7 Anforderungen an das prototypische Diagramm

Im Rahmen der Arbeit soll zur Demonstration der umgesetzten Konzepte ein vereinfachtes Diagramm erstellt werden. Dieses Diagramm wird in Anlehnung an ein Klassendiagramm der UML modelliert. Die grundlegenden Elemente von Klassendiagrammen sind in der Regel bekannt oder intuitiv erlernbar. Trotzdem besitzen Klassendiagramme durch Operationen, Attribute und verschiedene Verbindungstypen ein reichhaltiges Repertoire an Elementen, um die Möglichkeiten des Systems demonstrieren zu können. Das Diagramm soll aber nicht alle Funktionen eines Klassendiagramms nachstellen, sondern nur in Anlehnung an ein solches kreiert

3.6.8 Systemumgebung


3.7 Wunschkriterien


Im Rahmen dieser Arbeit werden nur Ressourcen betrachtet, die EMF und GMF in einer Datei speichern [vgl. 3.8]. Möglich ist allerdings die Betrachtung einer getrennten Speicherung beider Modelle.

Zusätzlich könnte in das System noch eine Chat-Funktionalität integriert werden, über welche die Nutzer direkt kommunizieren und Informationen zum aktuellen Entwicklungsstand austauschen könnten. Eine Auflistung aller Wunschkriterien findet sich unter [Anhang A]
3.8 Abgrenzungen


3.9 Anforderungskatalog

4 Konzeptentwicklung und Systemdesign

Nach der Analyse der Systemanforderungen legt das folgende Kapitel die konzeptuellen Grundlagen für die Implementierung. Es beschäftigt sich mit der Entwicklung von Lösungsansätzen der in Kapitel 3 geforderten Anforderungen an das System. Hierbei werden die Konzepte erarbeitet, welche die Implementierung und Umsetzung des Systems ermöglichen.

Basis jeder modellgetriebenen Entwicklung ist die Erstellung eines Prototyps, welcher als Muster für die Entwicklung der Generatoren fungiert. Aus diesem Grund werden zuerst alle Systemspezifika erläutert, die sich mit der Entwicklung und Konzeptionierung des Prototyps beschäftigen.


Im Anschluss verlagert sich der Fokus stärker auf den Server. Es werden die Persistierung der Daten, die Verwaltung des Systems und die Darstellung der Diagramme mittels Web-Viewer thematisiert. Ferner werden die Technologien und Konzepte für die Rechteverwaltung des Systems entwickelt und dargelegt.

Nachdem das System soweit konzeptioniert wurde, dass es durch eine Implementierung in einen lauffähigen Zustand überführt werden kann, wird in Kapitel 4.11 die modellgetriebene Generierung der einzelnen Komponenten veranschaulicht. Dies beinhaltet sowohl die Beschreibung der zu generierenden Komponenten, als auch des
sich darauf stützenden Datenmodells. Abbildung 25 gibt einen Überblick über die einzelnen Aufgaben, welche im Rahmen des Projektes zu bewältigen sind.

Abbildung 25 - Überblick über die einzelnen Sub-Projekte

Das System wurde, in Analogie zu sonnenbasierten Ereignissen (Eclipse = Sonnenfinsternis), auf den Namen Dawn (Dämmerung) getauft. Im Folgenden werden die Begriffe System und Dawn synonym verwendet.

4.1 Systemarchitektur


Das in Abbildung 26 dargestellte Verteilungsdiagramm erweitert den groben Systemüberblick aus Abbildung 24. Dabei agieren die einzelnen Komponenten ausschließlich über Schnittstellen, sind also folglich lose gekoppelt und austauschbar.
Die clientseitige Editor-Erweiterung wird grundlegend genutzt, um die spezifischen Besonderheiten jedes GMF-Editors an die Dawn-Runtime zu übertragen und diese mit den Eigenschaften des Editors zu initialisieren. Dabei nimmt die Erweiterung keinerlei Änderungen am bestehenden Editor vor, sondern ergänzt diesen nur um Informationen. Aus diesem Grund kann der GMF-Editor auch jederzeit losgelöst von Dawn genutzt werden. Im Kapitel 4.9 wird der Zusammenhang zwischen dem Editor-Plugin, der Erweiterung und der Dawn-Runtime detailliert dargestellt.


Der Server verwaltet alle Diagramm-Informationen in einer eigenen Einheit, die sich Projekt nennt. Projekte fügen jedem Diagramm serverseitig zusätzlich Meta-Informationen hinzu. So sind diese für die Betrachtung der Nutzer und Rechte und
ebenfalls für die serverseitige Speicherung der Diagramminformationen zuständig [vgl. Abbildung 27].

In der derzeit spezifizierten Version besteht eine eins-zu-eins Beziehung zwischen Diagramm und Projekt. Später kann aber durchaus ein Projekt für die Verwaltung mehrerer kontextzusammenhängender Diagramme genutzt werden. Ein Projekt stellt also den serverseitigen Container für die gesamte Behandlung von Diagrammen dar.


### 4.2 Kommunikationsstruktur

implementierungs-spezifischen Details des Protokolls offen. Dem gegenüber stehen performantere, nicht webbasierte Architekturen wie Java RMI. Um eine flexible Struktur zu schaffen, in welcher der Nutzer nicht auf eine einzige Technologie beschränkt ist, wurde ein generischer Ansatz für die Kommunikationsstruktur entwickelt – die Netzwerkadapter.

4.2.1 Netzwerkadapter


Abbildung 28 - System Design

Um zur Laufzeit die Remote-Connection clientseitig wechseln zu können, muss der Server alle zur Verfügung stehenden Adapter parallel anbieten. Der Server hält also alle Kommunikationskanäle zur Laufzeit offen. Dieses Design erlaubt es flexibel unterschiedlichste Kommunikationswege für die Clients zu ermöglichen. Dadurch können die Nutzer zeitgleich über die unterschiedlichsten Wege miteinander kommunizieren.


**4.2.2 Austausch der Ressource**

In Kapitel 2.8.4.5 wurde beschrieben, dass EMF- und GMF-Editoren ihr Datenmodell in Form von XMI abspeichern. Vorteil von XMI ist, dass es ein interoperables Austauschformat ist, mit dem auch andere Editoren fernab von Eclipse umgehen können. Dawn versteht sich zwar als kollaborative Erweiterung für GMF-Editoren, lässt
aber nicht die Vision einer viel stärker verteilten Architektur außer Acht. Aus diesem Grund wurde auch für die Übertragung des Modellbestandes eine nach XMI seriализierte Form gewählt. Dadurch kann es in späteren Entwicklungsstadien möglich sein, andere Editoren außerhalb der Eclipse-Welt mit in das System zu integrieren. Ein weiterer Grund für die Nutzung von XMI besteht in der Tatsache, dass es in XML spezifiziert, also textbasiert ist. Dies erlaubt eine einfache Integration in textbasierte Übertragungsmedien wie SOAP.


4.2.3 Watcher

Die wichtigste Schnittstelle zwischen der Übertragung der Daten und der Synchronisierung des Modellbestandes ist eine Systemkomponente, welche für den regelmäßigen Abruf der Daten sorgt. Diese Komponente wird von jeder

\textsuperscript{33} Lempel-Ziv-Welch, verlustfreies Komprimierungsverfahren, welches eine Code-Tabelle für die Komprimierung nutzt [vgl. Henning 2003, S.40]


\subsection*{4.3 Synchronisation}

Im Gegensatz dazu, wird das Übertragen der Daten vom Server an den Client als *Update* oder *Aktualisieren* bezeichnet. Hierbei übernimmt der Server die *Master*-Rolle. Alle Änderungen werden an den Client übertragen. Abbildung 29 verdeutlicht den Unterschied zwischen *Publish* und *Update*.

![Publish/Update Diagram](image)

Abbildung 29 - Publish/Update


### 4.3.1 Datenübertragung

Der offensichtlichste Ansatz, den Datenbestand der Clients mit dem Server zu synchronisieren, besteht darin die jeweiligen kompletten Datenbestände auszutauschen.


Abbildung 30 - Full-Sync-Delete Anomalie

Aktualisiert nun Client 1 seine Daten, bevor er diese speichert, so bekommt er vom Server nur Objekt A geschickt, und muss davon ausgehen, dass B gelöscht wurde. Infolgedessen entfernt er das Objekt ebenfalls aus seinem Datenbestand. Dieses Problem wurde Full-Sync-Delete Anomalie genannt, da die Löschung von Objekten aus
Konzeptentwicklung und Systemdesign

dem Modellbestand nur auftritt, wenn der komplette Datenbestand zur Synchronisation genutzt wird.


Um diese Problematiken wirksam zu umgehen, müssen gelöschte Objekte separat übertragen werden. Im optimalen Fall dürfen also nicht die kompletten Datenbestände, sondern nur die aktuellern Änderungen bezüglich Einfüge-, Änderungs- und Löschoperationen an den Server geschickt werden. Dies bewirkt, einhergehend mit der wirksamen Anomalie-Vermeidung, eine Steigerung der Performance. Auf das Performance-Verhalten des Systems wird detaillierter in 6.3 eingegangen.

4.3.2 Operationen

In dem System müssen Einfüge-, Änderungs- und Löschoperationen automatisch erkannt und verarbeitet werden können.

*Einfügeoperationen* stellen dabei die einfachste Operation dar. Sobald der Slave feststellt, dass sich innerhalb der Transport-Ressource Objekte befinden, die nicht lokal vorhanden sind, wird eine Einfügeoperation ausgelöst. Hierbei wird das Vorhandensein von Objekten anhand eines eindeutigen Identifikators geprüft [vgl. 4.3.3].

Für die Behandlung von *Änderungen* muss das System mehrere Parameter betrachten, um diese eindeutig identifizieren zu können. Wird bei der Überprüfung des eindeutigen

---

Identifikators festgestellt, dass ein Objekt sowohl in der Slave-Ressource, als auch in der Master-Ressource vorhanden ist, wird das Objekt auf Unterschiede geprüft. Da das System generisch für alle Objekt-Typen konzipiert ist, muss ein rekursiver Algorithmus alle Attribute und Kind-Objekte überprüfen [vgl. 4.3.3]. Da die Identifikation von Attributen und Kindern ebenfalls anhand eines Identifikators erfolgt, müssen Listen-Objekte in Master und Slave nicht gleich sortiert sein.

Auf Grund der Konfliktbehandlung [vgl. 4.4.4] kann das System davon ausgehen, dass Änderungen während eines Updates immer korrekt sind. Sie werden also direkt auf dem Server übernommen.


Löschoperationen ähneln in ihrer Behandlung den Einfüge-Operationen. Sobald innerhalb eines Publish-Vorgangs die Abwesenheit eines Objektes in der Ressource des Masters erkannt wird, wird das Objekt vom Server entfernt. Hierbei wird ebenfalls von der Richtigkeit der Client-Informationen ausgegangen. Lösch-Operationen auf dem Client, also während des Updates müssen gesondert behandelt werden, um zu vermeiden, dass lokal geänderte Objekte gelöscht werden [vgl. 4.4].

**4.3.3 Compare Engine**


4.3.4 Merge Engine


### 4.3.5 Schutz lokaler Änderungen


4.4 Konflikte


Da Dawn ein System ist, welches auf verteilten Ressourcen aufbaut und Änderungen nur in unregelmäßigen Abständen (Publish/Update) kommuniziert werden, erhöht sich die Gefahr von inkonsistenten Zuständen und Konflikten. Erschwert wird diese Problematik dadurch, dass der Server auf Grund der loser Kopplung keine Möglichkeit besitzt Änderungen an der Server-Ressource sofort an alle Clients zu kommunizieren. Er ist darauf angewiesen, in regelmäßigen Abständen den aktuellen Datenbestand abzulegen. Da ein Nutzer aber auch über einen längeren Zeitraum offline arbeiten können soll, müssen auch langläufige Konflikt-Potentiale erkannt und behandelt werden [vgl. 4.5].

konfliktverursachenden Situationen beschrieben, um anschließend Konzepte für die systemgesteuerte Erkennung dieser zu entwickeln. Daraufhin wird erläutert, wie erkannte Konflikte vom Nutzer behoben werden können. Anschließend werden Verfahren beschrieben, die es ermöglichen unter bestimmten Umständen Konflikte im System gänzlich zu vermeiden.

4.4.1 Konflikte – Client Seite

Konflikte können auf der Client-Seite auftreten, sobald der Nutzer eine Operation ausführt. Aus diesem Grund muss die Erkennung von Konflikten so schnell wie möglich erfolgen um zu verhindern, dass der Nutzer weiter mit einem nicht validen Modellzustand arbeitet und sich das Problem so ausweitet. Bevor Konflikte wirksam erkannt beziehungsweise vermieden werden können, muss ein System zuerst hinsichtlich möglicher Konfliktsituationen analysiert werden, um herauszufinden in welchen Situationen welche Konflikte auftreten können.

4.4.1.1 Lokal- und Remote-Änderungskonflikt


Der Objektzustand C wird als global gültig erklärt. An dieser Stelle ist ohne Gegenmaßnahmen das Entstehen eines Konfliktes nicht mehr zu vermeiden. Würde Client 1, wie in Schritt 4 dargestellt, seine Objektänderung publizieren, würden die Änderungen von Client 1 überschrieben werden. Das Szenario entspricht einer klassischen Race Condition\textsuperscript{35}, da Client 1 seine Daten publiziert ohne Kenntnis von den Änderungen anderer Clients zu erhalten.

\textsuperscript{35} Wettlaufsituationen in parallelen Systemen, die zu schwer auffindbaren Programmierfehlern führen können [vgl. Tanenbaum 2001, S.100ff.].
### 4.4.1.2 Lokaler Löschkonflikt


In Abbildung 32 führt Client 2 (entfernter Nutzer) eine Änderung an einem Objekt durch und verändert es so, dass es sich im Zustand B befindet.


### Abbildung 31 - Lokal- und Remote-Änderungskonflikt

<table>
<thead>
<tr>
<th></th>
<th>Client 1</th>
<th>Server</th>
<th>Client 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>publish</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>
4.4.1.3 Entfernter Löschkonflikt

Ein entfernter Löschkonflikt entsteht, ähnlich wie der lokale Löschkonflikt, wenn ein Nutzer ein Element löscht, an dem ein anderer Nutzer bereits Änderungen vorgenommen hat. Der Unterschied zum lokalen Löschkonflikt besteht darin, dass diese Löschung bei dem ändernden Nutzer noch nicht publiziert wurde. Er arbeitet folglich auf einem bereits gelöschten Objekt [vgl. Abbildung 33]. Um zu verhindern, dass das Objekt beim nächsten Veröffentlichen einfach wieder erstellt wird, muss der Nutzer über diesen Konflikt informiert werden, damit er ihn manuell beseitigen kann.

Prinzipiell stellt sich die Situation sehr ähnlich der unter 4.4.1.2 beschriebenen dar. Im Fall des entfernten Löschkonfliktes publiziert allerdings der Nutzer, welcher die Löschung ausgeführt hat zuerst seine Modellbestandsänderung. Würde nun Client 1 von Schritt 4 ausgehend seine Änderungen am Objekt publizieren, würde die Löschung von Client 2 rückgängig gemacht werden.
4.4.2 Konflikte – Serverseite


<table>
<thead>
<tr>
<th>Change Conflict Server-Site</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Client 1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Abbildung 34 - Änderungskonflikte serverseitig

muss also verhindert werden. Es setzt allerdings einen erhöhten Verwaltungsaufwand auf dem Server voraus.


Die Änderung von Nutzer 1 ist also eine direkte Folge seines aktuellen Bestandes. Der Server kann das Objekt in den neuen Zustand setzen. Zusätzlich erhöht er, da nun eine Veränderung an dem Objekt stattgefunden hat, die Version und informiert Nutzer 1 über diesen Wert (Schritt 4). Nutzer 2 möchte nun auch seine Änderungen publizieren, hat aber, wie zuvor, keine Notiz von dem neuen Objektzustand auf dem Server erhalten. Er


4.4.3 Konflikterkennung

Die in den obigen Kapiteln aufgeführten Konflikte müssen systemseitig zuverlässig erkannt werden, bevor sie behandelt werden können.


Neben der Funktion der Erkennung von Konflikten dient die LastResource auch als Vergleichsgrundlage, um lokale Änderungen zu identifizieren und diese bis zur nächsten Synchronisation zu registrieren. Der in Abbildung 36 abgebildete Algorithmus stellt dar, wie anhand der Objektvergleiche lokal analysiert wird, ob ein Konflikt aufgetreten ist, oder nicht.

Um Konflikte rechtzeitig erkennen zu können, wird diese Überprüfung mit jeder Aktualisierung durchgeführt. Diese Überprüfung wird ebenfalls durchgeführt, bevor der Client Daten an den Server publizieren möchte. Jedem Publish geht also ein Update voraus.

4.4.4 Konfliktbeseitigung

Konfliktfall nicht akzeptabel. Aus diesem Grund wird es in Dawn ermöglicht, dass der Nutzer selbst entscheiden kann, ob er die Version des Servers, also die letzte Änderung, akzeptiert oder sein Modell als valide einstuft. Diese Entscheidung kann der Nutzer für jedes in Konflikt getretene Objekt eigenständig treffen. Er ist also nicht daran gebunden sein gesamtes Modell zu publizieren oder zu verwerfen, sondern kann selektiv die Konflikte auflösen.

Abbildung 37 - Algorithmus zur Konfliktbehebung lokal geänderter Objekte
4.4.5 Konfliktvermeidung


Locking in Diagrammen ist ein sehr komplexer Vorgang. Es genügt nicht einfach nur ein Objekt zu locken, sondern es müssen sämtliche Referenzen zu diesem Objekt geprüft werden. Wird beispielsweise ein Lock auf eine Kante gesetzt, so müssen alle mit dieser Kante verbundenen Objekte mit gelockt werden, da in GMF-Editoren beim Löschen eines Knotens auch die entsprechenden Kanten verschwinden. Deshalb müssen verbundene Objekte mit reserviert werden, was zu komplexen Problemen führen kann, wenn andere Nutzer diese Objekte bereits reserviert haben. Um mehr Zeit in die Lösung der anderen Probleme investieren zu können, bereitet Dawn zwar die Basis für dieses komplexe Locking, setzt es aber nur für Knoten direkt um und belässt das Locking von Kanten vorerst in einem experimentellen Status.
4.5 Netzwerkunabhängigkeit


einen anderen Nutzer gelockt ist. Hierbei müssen sich die beiden Parteien über eine adäquate Konfliktlösung verständigen. Eine andere Möglichkeit diesem Problem zu begegnen, wäre das komplett Überschreiben aller Änderungen von gelockten Objekten, was aber unter Umständen einen erheblichen Datenverlust zur Folge hätte. Abbildung 38 stellt den Vorgang bei Wiederaufnahme der Verbindung grafisch dar.

Abbildung 38 - Flussdiagramm: Locking im Offline-Modus

Neben den Problemen bezüglich der Konfliktkennung und dem clientseitigen Locken von Objekten muss der Offline-Modus in der Lage sein das in Kapitel 4.6 behandelte Rechtesystem auch ohne Verbindung zum Server umsetzen zu können. Auch hierbei müssen die Informationen einen Neustart der IDE überstehen. Der Editor muss also, auch wenn er im Offline-Modus gestartet wird, sämtliche Rechte, die der Nutzer zu dem Zeitpunkt hatte, als die Netzwerkverbindung noch bestand, simulieren können, damit Nutzer nicht plötzlich in vollem Funktionsumfang das Modell editieren können. Alle drei Probleme (Konflikte, Locking und Rechte) an ihren spezifischen Stellen im Quellcode anpassen zu wollen, wäre vom Aufwand her und softwarearchitektonisch nicht tragbar. Aus diesem Grund wurde eine Komponente konzipiert, welche ausschließlich für die Verwaltung des Systems im verbindungslosen Betrieb zuständig


Abbildung 39 – Klassendiagramm: Adapter mit Offline-Klassen

### 4.6 Authentifizierung und Autorisierung


#### 4.6.1 Rollen und Operationen


Basis dieses generischen Nutzer- und Rechtesystems stellt hierbei ein Rollenkonzept dar, welches anhand von Rollen-Rechten die Zugriffrechte eines Nutzers innerhalb...
eines Projektes darstellt. Im Kern kann einem Nutzer in einem Projekt nur eine Rolle zugeordnet werden. Er kann sich aber innerhalb verschiedener Projekte in unterschiedlichen Rollen befinden, also unterschiedlich Rechte haben.


4.6.2 UserManager Konzept


Vorteil dieser Implementierung ist die Entkopplung der Projekte von der Nutzerverwaltung. Für jedes einzelne Projekt ist es transparent, welcher UserManager sich um die Rechte des Projektes kümmert, da es lediglich auf die entsprechende Instanz referenziert. Damit kann auch die Zuweisung der User-Manager zu den Projekten flexibel erfolgen. Beispielsweise kann jedes Projekt über seinen eigenen User-Manager verfügen [vgl. Abbildung 41, Fall 1]. Dadurch können die Rechte autark
vom restlichen System verwaltet werden. Es ist aber ebenso möglich für einzelne Gruppen von Projekten einen gemeinsamen UserManager zur Verfügung zu stellen [vgl. Abbildung 41, Fall 2]. Änderungen an dessen Rechten und Nutzern hätten somit direkte Auswirkung auf alle Projekte. Dieser Gedanke kann bis zu dem Punkt weitergeführt werden, dass alle Projekte einen gemeinsamen Nutzermanager referenzieren, der die gesamte Verwaltung übernimmt [vgl. Abbildung 41, Fall 3].

Abbildung 41 - Unterschiedliche Referenzierung von UserManagern

4.6.3 GlobalUserManager und NullUserManager


### 4.7 Datenbank und Persistenz


Abbildung 42 - ERM Diagramm des Persistenzmodells


36 http://xdoclet.sourceforge.net/


### 4.8 Webkomponenten


#### 4.8.1 Web-Viewer

4.8.1.1 Darstellung der Diagramme


Um GMF-Elemente in einem Browser darstellen zu können, muss eine grafische Entsprechung der einzelnen Views erstellt werden, wie sie der GMF-Entwickler spezifiziert hat. Da jeder GMF-Editor auf Java basiert, könnte ein Ansatz versuchen den Editor innerhalb eines Java-Applets im Browser ablaufen zu lassen. Allerdings würde, aufgrund der starken Abhängigkeiten zur Eclipse-Umgebung, das Applet schnell zu groß werden, um es einfach nutzen zu können. Dadurch würde die Performance stark reduziert und einhergehend damit ebenso die Akzeptanz der Nutzer. Außerdem setzt der Einsatz eines Java-Applets die Installation einer Java-Virtual-Maschine in der passenden Version auf dem Client voraus, was gerade für den Einsatz des Web-Viewers nicht der Fall sein soll, da dieser u.a. im Hinblick auf minimal konfigurierte Clients entwickelt wird. Die Nutzung von Java-Applets scheint folglich besser aus.

Ähnliche Argumente lassen sich gegen eine Implementierung mittels Flash anbringen. Auch hier muss der Browser in der Lage sein, die flashbasierten Animationen anzeigen zu können, was in jedem Fall voraussetzt, dass der Webbrowser das Flash-Plugin installieren und ausführen kann. Daneben besteht auch die Anforderung, dass die Web-Repräsentationen generisch aus dem GMF-Modell generiert werden können [vgl. 4.11]. Das proprietäre Format von Flash ist dabei denkbar ungünstig geeignet, um die Views aus einem Modell zu generieren.

Diese Überlegungen resultieren in die Nutzung von HTML in Verbindung mit JavaScript und CSS (Cascading Style Sheets) als Grundlage für den Web-Viewer. HTML und JavaScript sind textbasierte Formate, lassen sich also sehr leicht generieren. Außerdem bieten sie die nötige Performance und Systemunabhängigkeit. Durch Ajax (Asynchronous JavaScript and XML) als Kommunikationstechnologie ist es außerdem möglich mit einem Backend zu kommunizieren, was die spätere Erweiterung der

Implementierung zu einem Web-Editor ermöglicht. Einziger Nachteil von JavaScript ist die Teilweise nicht standardkonforme Implementierung in den einzelnen Browsern und die dadurch bedingte Erhöhung des Implementierungsaufwandes. Da einige Browser eigene Interpretationen oder Implementierungen von JavaScript nutzen (zum Beispiel der Internet Explorer mit JScript), kann es dazu kommen, dass bestimmte Systemteile für die einzelnen Browser angepasst werden müssen. Um dieses Problem im Rahmen dieser Arbeit zu vermeiden und den Fokus auf die reine Entwicklung legen zu können, werden alle Konzepte ausschließlich auf dem Firefox in der Version 3.x entwickelt.

Zur Darstellung der verschiedenen Kanten und Knoten, müssen diese Elemente mit denen in HTML zur Verfügung gestellten Mitteln gezeichnet werden. Leider sind diese Möglichkeiten sehr begrenzt, können aber durch die Nutzung von CSS durchaus erweitert werden, was aber meist mit einem erhöhtem Aufwand und einem nicht geringen Maß an Kreativität einhergehen muss. Auch muss für die spätere Nutzung als Editor mit bedacht werden, dass die einzelnen Elemente sich auf der browserbasierten Zeichenfläche verschieben lassen müssen. All diese Anforderungen selbst entwickeln und umsetzen zu wollen, würde den Rahmen dieser Arbeit sprengen. Nach eingehender Recherche fand sich ein Framework, welches einen großen Teil der geforderten Anforderungen bereits umsetzt. OpenJacob38 ist ein freies Framework zur Implementierung von Rich Internet Applications39. Ein Subprojekt von OpenJacob ist OpenJacob Draw2D, welches versucht Draw2D, die Zeichen-API von Eclipse [vgl. 2.8.5], mit Hilfe von Web-Technologien im Browser zu realisieren. Dabei können bereits komplexere Knoten und Kanten gezeichnet werden, was das API zu einer optimalen Grundlage für die eigene Entwicklung macht.

Für jedes View-Element eines GMF-Editors wird eine passende JavaScript-Klasse implementiert, welche dieselben Eigenschaften bezüglich der Darstellung aufweist wie die Java-Klasse innerhalb von Eclipse.


38 http://www.openjacob.org/


```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<figureMappings>
  <figureMapping type="2001">
    <javaScriptClass name="org.mftech.diagram.uml.clazz.AnAttributeLabelFigure" />
    <viewAttribute name="accessright" />
    <viewAttribute name="dataType" />
    <viewAttribute name="name" />
    <viewPattern name="{0} {2}:{1}" />
  </figureMapping>
</figureMappings>
```

Listing 3 - Mapping von ElementType zur JavaScript-View
Im aufgeführten Listing wird zuerst das erste Attribut, dann das dritte und zuletzt das zweite Attribut angezeigt. Hierbei werden die letzten beiden Attribute durch einen Doppelpunkt voneinander getrennt [Listing 3, Zeile 9]. Damit die Web-Darstellung und die Darstellung im Eclipse identisch sind, verarbeitet das Diagramm-Servlet zusätzlich diese Information und zeigt die Ausgabe richtig formatiert an.


Abbildung 43 - Web-Viewer Architektur


4.8.1.2 Kommunikation

dabei immer alle Informationen übertragen, egal ob sich Änderungen ergeben haben, oder nicht. Dies resultiert sowohl in einer erhöhten Last auf dem Server, der die Daten generieren muss, als auch in einer erhöhten Netzlast.


### 4.8.2 Web-Frontend


4.9 Dawn-Runtime und Editor-Erweiterung

Die Dawn-Runtime ist der elementare Kern der Client-Seite in Dawn. Sie kapselt alle Methoden, welche die Editoren zur Kommunikation mit dem Server benötigen. Daneben stellt sie Menüs und Dialoge zur Verfügung, um das System einfach und komfortabel verwalten zu können. Die Dawn-Runtime wird als eigenes Plugin realisiert, welches im Kontext von Eclipse läuft. Die Dawn-Erweiterung hingegen wird als Fragment [vgl. 2.8.3.4] implementiert. Dadurch werden alle zusätzlichen Erweiterungen im Kontext des Host-Plugins, also des GMF-Editors ausgeführt. Dies hat den Vorteil, dass am Host-Plugin keine Änderungen vorgenommen werden müssen, um Klassen erweitern zu können. Würde die Dawn-Erweiterung als Plugin implementiert, so müsste das GMF-Editor-Plugin alle zu erweiternden Klassen exportieren, also öffentlich zugänglich machen, was Änderungen am GMF-Editor Plugin bedeuten würde.

Neben den Wizards erweitert die Editor-Erweiterung auch den bestehenden Editor, ohne diesen zu verändern. Dies ist notwendig, um beispielsweise das Save-Event für die Synchronisation des Modellbestandes nutzen zu können.

![Abbildung 45 - Architektur des Clients](image)

Aus Abbildung 45 wird ersichtlich, welche Teile des GMF-Editors durch die Editor-Erweiterung erweitert werden. Jeder GMF-Editor erhält seine für ihn spezifische Erweiterung. Alle diese Erweiterungen greifen auf die Dawn-Runtime zu, um die Funktionalitäten von Dawn auszuführen. Dabei bietet die Dawn-Runtime die

4.9.1 Wizards


4.9.2 Preferences


<table>
<thead>
<tr>
<th>Name</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server URL</td>
<td>URL des Servers</td>
</tr>
<tr>
<td>Server Port</td>
<td>Port des Webservers</td>
</tr>
<tr>
<td>Project Name</td>
<td>Name des Web-Projektes</td>
</tr>
<tr>
<td>Publish Mode</td>
<td>Die Übertragungsmethode</td>
</tr>
<tr>
<td></td>
<td>• On Interval = Publish bei jedem Interval</td>
</tr>
<tr>
<td></td>
<td>• On Save = Publish nur beim Speichern</td>
</tr>
<tr>
<td>Connection Mode</td>
<td>Die Verbindungsmethode, beziehungsweise das Übertragungsprotokoll. Zur Zeit SOAP oder RMI</td>
</tr>
</tbody>
</table>

Tabelle 4 - Preferences

4.9.3 Dawn Conflict View

Nicht alle Konflikte können direkt über das Diagramm gelöst beziehungsweise angezeigt werden. Lokale Lösch-Konflikte [vgl. 4.4.1.2] beziehen sich auf Objekte, die lokal gelöscht, aber von einem anderen Clients geändert wurden. Da sich diese Objekte
Konzeptentwicklung und Systemdesign  


4.10 Konzept des prototypischen Diagramms


Abbildung 46 - prototypisches Klassendiagramm
Da jeder GMF-Editor auf einem Ecore-Modell basiert, musste ein Datenmodell für den Editor entworfen werden, das die Elemente wie Klasse, Interface, Assoziation etc. bestimmt. Da die Bezeichner des Modells in englischer Sprache verfasst sind, Elemente wie *interface* oder *class* aber reservierte Schlüsselwörter in Java sind, wurden alle Bezeichner mit dem Prefix „A“ oder „An“ versehen. Abbildung 47 zeigt Ecore-Modell für den Prototyp.

**4.11 Code-Generierung**

Die vorherigen Kapitel legten den konzeptuellen Grundstein für die Erstellung eines Prototyps für das gesamte Dawn-System. In der Regel ist dies einer der ersten Schritte im modellgetriebenen Softwareentwicklungsprozess. Auf dem Prototyp aufbauend können, so noch nicht geschehen, Modelle entwickelt werden, welche durch Generatoren transformiert werden können.

**4.11.1 Dawn-GenModel**

4.11.2 Generierung der Komponenten


Verfügung gestellt, damit der Server direkt benutzt werden kann. Natürlich verhindert die Nutzung des Tomcat-Plugins nicht, dass der Server auch in anderen Application-Servern genutzt werden kann.


4.12 Zusammenfassung


Dawn nutzt ein flexibles Kommunikations-Adapter-Konzept, um die Übertragungstechnologie für die Daten austauschen zu können. Dieser Wechsel kann sogar zur Laufzeit erfolgen [vgl. 4.2].


Um die Konflikte zu beseitigen, nutzt Dawn ein offenes System, was es dem Nutzer erlaubt den Konflikt sowohl durch Annahme, als auch durch Verweigerung der Änderungen des anderen Nutzers zu lösen. Damit Konflikte ganz ausgeschlossen werden können, kann das System bestimmte Bereiche eines Diagramms blockieren [vgl. 4.4].

Dawn unterstützt das Bearbeiten des Diagramms, auch wenn keine Verbindung zum Server besteht. Es nutzt hierzu eine Offline-Server Instanz, die sich bei bestehender Verbindung mit dem Dawn-Server synchronisiert und im Bedarfsfall den Server simulieren kann [vgl. 4.5].

Über ein generisches Rechte- und Rollenkonzept ist das System so flexibel, dass spätere Anforderungen hinsichtlich der Vergabe von Rechten einfach implementiert werden können. Im Zentrum dieses Konzeptes steht die Komponente UserManager, an die alle nutzerspezifischen Anfragen weitergeleitet werden. Durch spezielle Implementierungen dieser Schnittstelle können die Fähigkeiten individuell gestaltet werden, sodass es auch möglich ist das Nutzermanagement abzuschalten ohne Einfluss auf andere Komponenten nehmen zu müssen [vgl. 4.6].

Um die GMF-Diagramme auch in einem Browser anzeigen zu können, wurde ein Konzept entworfen, dass die Figuren mit Hilfe von HTML und JavaScript angezeigt werden können. Dabei übernimmt das *DiagramServlet* die Rolle des Controllers und kann mit Hilfe generierter Konfigurationsinformationen eine Zuordnung zwischen den Views und dem persistierten Datenmodell herstellen [vgl. 4.8.1].

Der Server, sowie die JavaScript-Views und die Editor-Erweitung können mittels eines modellgetriebenen Prozesses erzeugt werden. Dabei basiert die Generierung auf dem Dawn-GenModel. Dieses greift bei der Generierung auf die bestehenden GMF-Modelle zu und kann so deren Einstellung in die Erweiterungen einfließen lassen[vgl. 4.11].
5 Implementierung


5.1 Kommunikation und Synchronisation

In den Kapiteln 4.2 und 4.3 wurden die Konzepte für die Kommunikation und Synchronisation in Dawn erläutert. Nachfolgend werden einige interessante Aspekte der Implementierung beschrieben.

5.1.1 Initialisierung der Kommunikation


Beim Start von Eclipse werden alle Plugins gestartet und alle bereits offenen Editoren initialisiert. Hat also ein Benutzer Eclipse geschlossen und alle Diagramme, an denen er in der aktuellen Sitzung gearbeitet hat, offen gelassen (was dem Normalfall entspricht), so werden diese sofort beim nächsten Start erzeugt. Dies hat zur Folge, dass während die Eclipse-Instanz startet der Vorgang unterbrochen wird und über dem Startbild ein Login-Dialog angezeigt wird. Folgerichtig muss das Starten der Wächter-Threads solange verzögert werden, bis die Eclipse GUI vollständig geladen wurde. Zwar bietet Eclipse verschiedene Methoden an, welche den Zustand der Workbench anzeigen (z.B.

Mit Hilfe eines kleinen Tricks kann das Problem allerdings dennoch behoben werden. Der Unterschied zwischen dem gestarteten und dem startenden Eclipse ist, dass verschiedene GUI-Komponenten erst nach dem Start initialisiert sind. So ist zum Beispiel die Menu-Bar nur dann initialisiert, wenn die Eclipse-Workbench sichtbar ist. Mit Hilfe dieser Information kann also das Starten von Komponenten solange verzögert werden, bis die Menu-Bar vom System erzeugt wurde. Listing 4 zeigt die Quelltext-Passage, mit der das zeitverzögerte Starten der Komponenten realisiert wird.

```
while (Display.getCurrent().getActiveShell().getMenuBar() == null)
{
    System.out.println("Eclipse has not finished starting...wait");
    try
    {
        Thread.sleep(200);
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
```

Listing 4 - Zeitverzögertes Starten der Dawn Komponenten

### 5.1.2 RemoteConnections und RemoteConnector

Implementierung vom Typ Object (Object[]). Um die eigentliche Funktionalität aber hinter dem RemoteConnection-Interface verbergen zu können, wurde eine WrapperKlasse geschrieben, welche die Methoden des generierten SOAP-Stubs kapselt und gegebenenfalls die Object-Arrays in die vom Interface erwarteten Listen konvertiert.


```java
if (currentType == RMI) {
    mfInstance = (DawnRemoteConnectionRMI) Naming.lookup("rmi://" +
    servername + "/" + serverContextname);
}
else if (currentType == SOAP) {
    String serverAdress = "http://" + serverName + ":" + serverport +
    "/" + serverContextname + "/services/DawnCommunicationService";
    mfInstance = new DawnRemoteConnectionSOAP(serverAdress);
}
```

Listing 5 - Auswahl des Kommunikationsprotokolls

Die einzelnen serverseitigen Adapter greifen alle direkt auf die Singleton\textsuperscript{40}-Instanz des Servers zu. Um zu verhindern, dass gleichzeitiger Zugriff auf die Daten zu Inkonsistenzen führt, sind alle Methoden des Servers gegen parallele, mehrfache Nutzung geschützt. Dies wurde umgesetzt, indem die kritischen Bereiche der einzelnen Methoden mit synchronized-Blöcken versehen wurden. Diese Bereiche werden von der Java Virtual Machine geschützt ausgeführt.

5.2 Umsetzung des prototypischen GMF-Editors

Der an ein Klassendiagramm angelehnte GMF-Editor wurde basierend auf dem spezifizierten EMF-Modell [vgl. 4.10] und der grafischen Vorlage [vgl. Abbildung 46] implementiert. Dabei wurden dem GMF-Entwicklungsprozess entsprechend erst die grafischen Repräsentationen (Figures) und die Tooling-Palette definiert und diese daraufhin mit dem Datenmodell verknüpft. Über die modellgetriebenen Mechanismen von EMF und GMF wurden daraufhin die Datenhaltungsklassen, das Edit-Plugin und

\textsuperscript{40}Singleton – Entwurfsmuster, bei dem sichergestellt wird, dass von einer Klasse nur eine Instanz erzeugt werden kann [vgl. Schmidt 2007, S.102]
Implementierung


5.3 Editor-Erweiterung


5.3.1 ElementTypeHelper und ResourceSet


```java
public class ClassdiagramResourceSet {
    public static ResourceSet getResourceSet() {
        Map m = reg.getExtensionToFactoryMap();
        m.put("*", new XMIResourceFactoryImpl());

        ResourceSet rsSet = new ResourceSetImpl();
        rsSet.setResourceFactoryRegistry(reg);
        rsSet.setPackageRegistry()
            .put("http://www.eclipse.org/gmf/runtime/1.0.1/notation",
                NotationPackage.eINSTANCE);
        rsSet.setPackageRegistry()
            .put("http://www.Classdiagrameditor.de",
                ClassdiagramPackage.eINSTANCE);
        return rsSet;
    }
}
```

Listing 6 - editorspezifisches ResourceSet
In Listing 6 wird in den Zeilen 12-17 diese Registrierung vorgenommen. Jeder Editor benötigt dieses ResourceSet, um es der Dawn-Runtime zur Laufzeit übergeben zu können.

Neben dem ResourceSet benötigt jeder Editor eine Klasse, welche es ermöglicht eine Zuordnung zwischen einem View-Element und seinem ElementType zu treffen. Leider bietet das GMF-Framework keine Möglichkeit direkt an diese Information zu kommen, weshalb eine eigene Implementierung erstellt werden musste. Zwar werden die ElementTypes in der ElementTypeRegistry eines Editors verwaltet, können dort aber nur über einen statischen String abgefragt werden. Das generische Mapping zu einer View kann nicht erfolgen. Die Komponente, der ElementTypeHelper, erstellt die Zuordnung, indem er die VisualID einer View mit der eines passenden EditParts vergleicht, um den entsprechenden ElementType zu finden. Listing 7 zeigt einen Ausschnitt des ElementTypeHelpers für den prototypischen Klassendiagramm-Editor.

```java
public IElementType getElementType(View view)
{
    int visualId = ClassdiagramVisualIDRegistry.getVisualID(view);
    switch (visualId)
    {
        case ClassDiagramEditPart.VISUAL_ID :
            return ClassdiagramElementTypes.ClassDiagram_79;
        case AnAttributeEditPart.VISUAL_ID :
            return ClassdiagramElementTypes.AnAttribute_2001;
        default :
            return null;
    }
}
```

Listing 7 - getElementType(View)

Damit die einzelnen Implementierungen einfach von den Anwendern erweitert werden können, arbeitet die Dawn-Runtime nicht direkt mit diesen Implementierungen, sondern mit Fabriken, welche die Erzeugung des ResourceSets und des ElementTypeHelpers übernehmen. Welchen Mechanismus die Dawn-Runtime nutzt, um zur Laufzeit die dynamischen Informationen der einzelnen Editoren zu erhalten, wird in Kapitel 5.4.4 beschrieben. Kapitel 5.9 beschäftigt sich mit den generativen Aspekten dieser beiden Komponenten.

### 5.3.2 DawnExtendedEditor

Kern eines jeden GMF-Editors ist eine Klasse, welche die Basis-Klasse DocumentDiagramEditor erweitert. Diese Editor-Klasse verwaltet nicht nur Lade- und


![Abbildung 49 - Dawn Information](image_url)

Um allerdings bei Erweiterung bzw. Regenerierung des GMF-Editors oder bei Erweiterungen seitens des GMF-Editor-Entwicklers keine Konflikte zu erzeugen, wird dieser Menüpunkt immer an die letzte Stelle in der Menüleiste gesetzt [vgl. Abbildung...
Bei der Erstellung der GUI greift der DawnExtendedEditor dabei auf die RootPalette zu und fügt die entsprechenden Einträge hinzu. Hierzu überschreibt er die Methode zur Erstellung der RootPalette (PaletteRoot createPaletteRoot (PaletteRoot existingPaletteRoot)) des eigentlichen Editors.

Um die Anzeige der Serververbindung ändern zu können, stellt der DawnExtendedEditor eine öffentliche Methode bereit (void showServerOnlineMode (boolean)). Diese Methode nutzt die Watcher-Implementierung für den Fall, dass die Verbindung zum Server nicht hergestellt werden kann, um den Editor zu veranlassen das Symbol in dem GUI zu verändern.


```java
public interface DawnDiagramEditorInterface
{
    public Watcher getWatcher();
    public void setOwnPaletteVisible(boolean visible);
    public String getContributorID();
    void showServerOnlineMode(boolean online);
}
```

**Listing 8 - DawnDiagramEditorInterface**

Sämtliche Konzepte für die Veränderung des eigentlichen Editors basieren ausschließlich auf dessen Erweiterung. Somit ist es ohne Probleme möglich den GMF-Editor losgelöst von Dawn zu betreiben.
5.3.3 Wizards


Abbildung 50 - Wizards zum Erstellen und Beitreten von Projekten
5.4 Dawn-Runtime

Die Dawn-Runtime ist die zentrale Komponente, um die Funktionalitäten von Dawn auf der Clientseite umzusetzen. In den folgenden Kapiteln wird auf einige interessante Aspekte ihrer Implementierung eingegangen.

5.4.1 Konfliktbehandlung und -behebung


Abbildung 51 - Anlegen der Action für die Konfliktbehandlung


![Abbildung 52 - Editor mit Konflikten](image.png)

### 5.4.2 Dawn Conflict View

Nicht alle Konflikte können allerdings im grafischen Editor angezeigt werden. Objekte, welche lokal gelöscht wurden, müssen anderweitig als Konflikt vermerkt und gelöst
werden können. Hierzu bietet es sich an die von Eclipse bereitgestellten Views zu verwenden.


Da die Conflict-View eine baumbasierte Repräsentation zur Darstellung der Konflikte benutzt, wird die JFace-Klasse TreeViewer als Grundlage für die Darstellung genutzt. Wie bei allen JFace-Elementen, liefern ContentProvider die Daten für die Anzeige. Dadurch wird eine strikte Trennung zwischen View und Modell erreicht. Außerdem bereiten die ContentProvider die Daten für ihre entsprechende View auf. So besitzen baumbasierte Ansichten andere ContentProvider-Implementierungen, als beispielsweise tabellarische Ansichten. Wie bei JFace-Viewern üblich wurde der ContentProvider als innere Klasse realisiert. Diese Implementierung kann über den aktuellen Editor auf den ResourceSynchronizer zugreifen und dort die aktuell in Konflikt stehenden Objekte abfragen. Nachdem der ContentProvider diese Informationen erhalten hat, kann er sie in das für die Conflict-View benötigte Format umwandeln und über den Viewer zur Anzeige bringen.

Abbildung 53 - Dawn Conflict View

In der View werden, sortiert nach den drei Konflikten, alle betroffenen Objekte angezeigt [vgl. Abbildung 53]. Für jedes Objekt steht das bereits beschriebene Kontext-Menü zur Verfügung, um den Konflikt aufzulösen.

5.4.3 Locking


Bei jedem Polling-Intervall wird als erstes vom Client diese Liste vom Server geladen. Alle Objekte, die nicht vom lokalen Nutzer gelocked wurden, werden durch eine gelbe Umrandung markiert und können nicht mehr editiert werden [vgl. Abbildung 54, links].

Die gelockten Objekte werden sofort eingefroren. Sobald der Nutzer, der den Lock auf ein Objekt gesetzt hat, eine Änderung publiziert, wird diese auf allen anderen Clients verbreitet, unabhängig davon ob bereits Änderungen an dem Objekt ausgeführt wurden.

Abbildung 54 - Locken eines Objektes

5.4.4 Extentions Points und Dawn Extension Service

bereits instanziert werden, obwohl sie noch nicht benötigt werden. Zum anderen ist auch der Umgang mit Singletons in Eclipse sparsam zu nutzen.

Eine weitere und wesentlich elegantere, und deshalb auch eingesetzte Lösung, ist die Nutzung von Extension Points zur Registrierung der einzelnen Systemteile.

Um von einem Plugin aus einen Extension Point zur Verfügung stellen zu können, wird eine XML-Schema Definition (*.exsd) erstellt, in der beschrieben wird, welche Parameter die Erweiterung anbieten soll und unter welchem Namen sie am Plugin gefunden werden kann. Hierfür stellt Eclipse einen Editor bereit, mit dem diese Einstellungen komfortabel vorgenommen werden können [vgl. Abbildung 55].

Für die Dawn-Runtime wurden zwei Erweiterungspunkte definiert. Einer für die Nutzung des ElementTypeHelpers und einer für das ResourceSet des jeweiligen Editors. Um die Komponenten aber einfach trennen zu können und es dem clientseitigen Entwickler zu erlauben, einfach die konkreten Implementierungen auszutauschen, werden die Komponenten hinter Schnittstellen verborgen, deren konkrete Implementierung durch Fabriken erzeugt wird. An den Erweiterungspunkten werden die Fabriken der jeweiligen Komponente registriert.


```java
public DawnElementTypeHelperFactory
createDawnElementTypeHelperFactory(String pluginId)
{
    IConfigurationElement[] config =
    Platform.getExtensionRegistry()
    .getConfigurationElementsFor(DAWN_FACTORY_ID);
    for (IConfigurationElement e : config)
    {
        Object o = e.createExecutableExtension("class");
        if (o instanceof DawnElementTypeHelperFactory)
        {
            if (e.getNamespaceIdentifier().equals(pluginId))
            {
                return (DawnElementTypeHelperFactory) o;
            }
        }
    }
    return null;
}
```

Listing 9 - Dawn Extension Service

5.4.5 Preferences


Um diese Einstellungen initialisiert und mit Default-Werten belegt werden können, muss der zweite Extension Point erweitert werden. Über die Erweiterung org.eclipse.core.runtime.preferences wird eine weitere Klasse in das System eingebunden, die die Initialisierung der Daten übernimmt. Diese Klasse (PreferenceInitializer) erweitert die abstrakte Basisklasse
Die Preferences sind global verfügbar und werden von dem Dawn-Runtime Plugin zur Verfügung gestellt.

5.4.6 Offline-Server


5.5 Persistenz

Hibernate abstrahiert den Zugriff auf das darunter liegende Datenbankbetriebssystem und bietet eine Schicht für das Umsetzen von objektorientierten Objekten auf relationale Tabellen. Um den Zugriff mittels Hibernate aber weiter zu abstrahieren und dadurch auch eine Unabhängigkeit von der objektrelationalen Implementierung zu bekommen, stellt Dawn eine Zwischenschicht zur Verfügung - den DBManager. Für jede Klasse die
Implementierung

in der Datenbank gespeichert werden soll, liefert der DBManager verschiedene, auf die Klassen zugeschnittene Funktionen an, um die Objekte laden, speichern oder löschen zu können [vgl. Abbildung 57].

Um diese Methoden nicht von Hand generieren zu müssen, wurde eine einfache Generator-Klasse geschrieben, welche, mit den entsprechenden Klassen instanziert, den DBManager erstellen kann. Dieser simple Generator nennt sich DBManagerCreator.

```java
public class HibernateCreationUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            Configuration configuration = new AnnotationConfiguration();
            Configuration.setProperty("hibernate.hbm2ddl.auto", "create-drop");
            sessionFactory = configuration.configure().buildSessionFactory();
        } catch (Throwable ex) {
            ...}
    }

    catch (Throwable ex) {
        ...}
}
```

Hibernate kann so konfiguriert werden, dass es nicht nur die Daten in die Datenbank überträgt, sondern diese auch erstellt. Dazu muss in der globalen Konfiguration das Attribut hibernate.hbm2ddl.auto auf den Wert create-drop gesetzt werden. Dies kann sowohl in der Konfigurations-XML als auch programmtechnisch geschehen.

### 5.6 Web-Viewer

Kern-Komponente für das Anzeigen der Modell-Informationen des Servers in einem Browser ist das unter 4.8.1 beschriebene Diagramm-Servlet, welches die Modell-Informationen vom Dawn-Server geliefert bekommt und diese als HTML/JavaScript Figuren ausgibt [vgl. 4.8.1].

Das Problem bei der Anzeige der View ist, dass das Diagramm-Servlet lediglich die Modellinformationen, also die reinen Daten des Modells, sowie Modelldaten der Figuren vom Dawn-Server geliefert bekommt. Das Servlet hat also Kenntnis darüber, wo sich die Knoten und Kanten befinden, und welche Größe sie haben, aber keine weiteren Informationen über deren Aussehen. Auf der Clientseite übernimmt das Layout eine generierte Figure-Klasse, welche in den EditPart eingebettet ist. Auf der Serverseite werden JavaScript-Figuren für die Anzeige genutzt, welche in Dawn basierend auf dem GMF-GenModel generiert wurden [vgl.5.9.3].


Da der Server viele verschiedene Editoren verwalten kann und demzufolge verschiedene JavaScript-Pakete unterstützen muss, werden alle editoreigenen Web-Daten in einem Ordner zusammengefasst, der als Namen die Paketbezeichnung des Editors trägt. Der prototypische GMF-Editor würde beispielsweise in dem Verzeichnis org.mftech.diagram.uml.class.diagram abgelegt werden. Dieses Vorgehen dient nicht nur der Separierung der einzelnen Editoren, sondern wird auch als Identifikator
Implementierung  143

genutzt, damit das Diagramm-Servlet die richtigen Repräsentationen für den Editor anzeigen kann. Jedes dieser Verzeichnisse enthält einen Ordner für die Meta-Daten (meta), welches die config.xml enthält und in dem bei Bedarf weitere Meta-Daten untergebracht werden können. Zusätzlich befindet sich unter diesem Root-Verzeichnis das Verzeichnis icons, das die Bild-Dateien des GMF-Editors beinhaltet [vgl. Abbildung 58, rechts].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="figureMappings">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="figureMapping"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="figureMapping">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="javaScriptClass"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="viewAttribute"/>
        <xs:element minOccurs="0" ref="viewPattern"/>
      </xs:sequence>
      <xs:attribute name="type" use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="javaScriptClass">
    <xs:complexType>
      <xs:attribute name="name" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="viewAttribute">
    <xs:complexType>
      <xs:attribute name="name" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="viewPattern">
    <xs:complexType>
      <xs:attribute name="name" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

**Listing 12 - config.xml Schema**

Die zu erstellenden JavaScript-Views unterteilen sich in vier Kategorien – Knoten, Kanten, Kanten-Dekoratoren und Label. Um die spätere Generierung und die Nutzung beziehungsweise die Erweiterung der JavaScript-View zu vereinfachen, wurde für jede
Implementierung 144
dieser Repräsentationen eine Basis-Klasse erschaffen, von der die konkreten
Ausprägungen das grobe Aussehen erben und über Parameter erweitern können. Diese
Basis-Klassen wurden ebenfalls in einem speziellen Ordner abgelegt
(org.mftech.dawn.basic). Abbildung 58 zeigt wie die Informationen aus dem GMF-
Modell (links) in Java-Script Klassen (rechts) umgewandelt wurden.

<table>
<thead>
<tr>
<th>GMF-Graph</th>
<th>JavaScript Figures</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="#">Diagramm mit GMF-Modell</a></td>
<td><a href="#">JavaScript-Views</a></td>
</tr>
</tbody>
</table>

Abbildung 58 - Zuordnung GMF-Modell zu JavaScript-Views

Das Diagramm-Servlet generiert den kompletten JavaScript-Quelltext für die Anzeige
eines bestimmten Diagramms. Eine Schwierigkeit bei der Erstellung des Quellcodes
war es, die Variablenbezeichner dynamisch zu erstellen. Jeder Knoten und jede Kante
musste einen eindeutigen Variablennamen besitzen, damit der Quelltext syntaktisch
corrckt ist. Um dies zu bewerkstelligen, wurden die XMI-IDs, welche jede View
eindeutig beschreibt, als Variablennamen verwendet. Leider können XMI-IDs
syntaktische Eigenheiten aufweisen, die es verhindern, sie als gültige JavaScript-
Variablen zu nutzen. Beispielsweise können XMI-IDs das Minus-Zeichen enthalten.
Um diese Unzulänglichkeit auszubessern, wurden mittels eines Konverters die XMI-IDs
in einen gültigen String umgewandelt. Zum besseren Verständnis der Funktionsweise
des Diagramm-Servlets ist in Listing 13 ein einfaches Modell (semantic und notational),
wie der Server es speichert, dargestellt. Listing 14 und Listing 15 zeigen den generierten
JavaScript-Quelltext, wie er auf dem Client angezeigt wird, und die config.xml für die
Zuordnungen. Der Übersicht halber wurden Deklarationen und sonstige nicht benötigte
Inhalte weggelassen. Wichtige Aspekte sind in dem Listing fett dargestellt.

Zeile 1 [Listing 14] erstellt die Zeichenfläche, also das Diagramm. In den Zeilen 2-3
wird ein Knoten vom Typ `AClassFigure` instanziert. Hierbei wird als
Variablenbezeichner die umgewandelte XMI-ID aus dem Notational-Modell verwendet [Listing 13, Zeile 20; Listing 14, Zeile 5]. Über den Type-Identifikator kann das Diagramm-Servlet mit Hilfe der config.xml [Listing 15, Zeilen 3-5] die Zuordnung zur JavaScript-Figure AClassFigure treffen. Da dieser Knoten ein Label enthält (Type 4002), wird dieser ebenfalls erzeugt und dem Knoten hinzugefügt [Listing 14, Zeile 5]. Der Wert, der in dem Label angezeigt werden soll, ist das Attribut name, welches sich im semantischen Modell der Klasse befindet [Listing 13, Zeile 6].

Listing 13 - vereinfachtes Modell (semantic und notational)

Diese Information wird ebenfalls aus der config.xml [Listing 15, Zeile 13] gewonnen. Wie bereits beschrieben, wird der eigentliche Wert des Attributes mit Hilfe von
Implementierung

Dynamic EMF ermittelt. Die Layout-Informationen des Knoten werden aus dem Notational-Modell gewonnen [Listing 13, Zeile 36] und die JavaScript-Figure damit parametrisiert [Listing 14, Zeile 17-18].


[Abbildung 59 - GMF- und JavaScript-Knoten im Vergleich]

Mit Hilfe des Web-Viewers können aber nicht nur einfache viereckige Knoten dargestellt werden, sondern auch verschiedenste Kanten, Knoten mit runden Ecken oder Konnektoren an den Kantenenden. Abbildung 60 stellt einen Vergleich zwischen einem Diagramm in dem prototypischen Editor (rechts) und der entsprechenden Darstellung im Browser (links) dar. In Kapitel 5.9.3 wird beschrieben wie mit Hilfe des GMF-Graph-Modells das Aussehen der JavaScript-Figuren beeinflusst werden kann.
var workflow = new draw2d.Workflow(""); 
var _pxDwwfLEEd2tPuftDUWv9g = new org.mftech.diagram.uml.clazz.AClassFigure(); 

var _pxDwylPLEEd2tPuftDUWv9g_0 = new org.mftech.diagram.uml.clazz.AClassNameFigure("Klasse"); 
_pxDwwfLEEd2tPuftDUWv9g.addChild(_pxDwylPLEEd2tPuftDUWv9g_0); 

var _pxDwyPLEEd2tPuftDUWv9g_2 = new org.mftech.dawn.basic.DawnCompartmentFigure(""); 
var _4UTzgB4hEd69o8NIisUctMg_0 = new org.mftech.diagram.uml.clazz.AnOperationLabelFigure("public foo()"); 
_pxDwyPLEEd2tPuftDUWv9g_2.addChild(_4UTzgB4hEd69o8NIisUctMg_0); 
_pxDwwfLEEd2tPuftDUWv9g.addChild(_pxDwyPLEEd2tPuftDUWv9g_2); 
_pxDwwfLEEd2tPuftDUWv9g.setDimension(86, 46); 
workflow.addFigure(_pxDwwfLEEd2tPuftDUWv9g, 115, 190); 

Listing 14 - Ausschnitt - erzeugter Diagram Quelltext

<?xml version="1.0" encoding="ISO-8859-1"?>
<figureMappings>
  <figureMapping type="1002">
    <javaScriptClass
      name="org.mftech.diagram.uml.clazz.AClassFigure" />
  </figureMapping>
  <figureMapping type="4002">
    <javaScriptClass
      name="org.mftech.diagram.uml.clazz.AClassNameFigure" />
    <viewAttribute name="name" />
    <viewPattern name="" />
  </figureMapping>
  <figureMapping type="2004">
    <javaScriptClass
      name="org.mftech.diagram.uml.clazz.AnOperationLabelFigure" />
    <viewAttribute name="accessright" />
    <viewAttribute name="dataType" />
    <viewAttribute name="name" />
    <viewPattern name="{0} {2}():{1}" />
  </figureMapping>
  <figureMapping type="5004">
    <javaScriptClass
      name="org.mftech.dawn.basic.DawnCompartmentFigure" />
  </figureMapping>
</figureMappings>

Listing 15 - exemplarische config.xml
Abbildung 60 - Web-Viewer (links) und GMF-Editor (rechts)


Um die Darstellungsschicht von den dynamischen Elementen innerhalb des Diagramm-Servlets zu trennen, wurden die statischen Bestandteile des Servlets in eine JSP (Java Server Page) ausgelagert. Hierzu zählen sowohl die strukturgebenden Tags wie <html>, <head> oder <body>, als auch die Include-Anweisungen, die von allen Diagrammen genutzt werden. Die dynamischen Aspekte, also die Diagrammspezifischen Inkludierungen und das eigentliche Diagramm, wurden gesondert generiert. Um diese ebenfalls in den Kontext der JSP einbetten zu können, wurde die dynamische Generierung in eigens dafür erstellte Tags ausgelagert. Hierzu wurde eine Tag-Bibliothek erstellt, über welche die Tags zugreifbar gemacht wurden. Listing 16 stellt auszugsweise die Web-Viewer-JSP und den Zugriff auch die dynamischen Aspekte dar.

41 http://www.prototypejs.org/
5.7 Web Konfigurations-GUI und Projektverwaltung


Für einen Benutzer, der gerade in der Eclipse-IDE ein Projekt angelegt hat, wäre es aber umständlich in einen Browser zu wechseln, sich einzuloggen, das eben erstellte Projekt auszuwählen und dort die Änderungen vorzunehmen.


```plaintext
http://localhost:8080/DawnServer/configureProject.do?projectname=default15.umleditor_diagram&username=Martin&token=1230663014218&hash=f4fe8a777684615f94b398ddf9311c0d3574b38
```
Listing 17 - Beispiel-Anfrage mittels SSO

5.8 Umsetzung des Rechte-Konzepts

Clientseitig fragt die Dawn-Runtime die Nutzer-Rechte vom Server ab. Hierzu muss der Nutzer sich authentifizieren. Dies geschieht, sobald der Editor gestartet wird und eine Verbindung zum Server hergestellt werden kann. Um zu verhindern, dass Änderungen am Diagramm vorgenommen und publiziert werden können, ohne sich authentifiziert zu haben, wird als erstes ein Login-Dialog angezeigt, der das Diagramm blockiert. Erst wenn der Nutzer authentifiziert und für das Diagramm autorisiert wurde, kann er Änderungen vornehmen. Sind die Nutzerinformationen validiert, werden sie in einer lokalen Session (LocalSession) hinterlegt. Die LocalSession ist als Singleton implementiert und kann beliebige Objekte unter einem Identifikator speichern. Sie kann also neben den Nutzereinstellungen auch noch andere Informationen verwalten und dem System global zur Verfügung stellen. Mit jeder update-Operation werden die aktuellen Rechte, die auch dem Server für das gerade geöffnete Diagramm hinterlegt sind, zum
Client übertragen. Haben sich Änderungen ergeben, so werden diese sofort umgesetzt. Wird dem Nutzer beispielsweise das Recht auf Änderungen entzogen, so wird das Diagramm gesperrt und kann nur noch betrachtet werden.

5.9 Dawn-Codegen


**5.9.1 Generierung des Servers**

Der Creator für die Erzeugung des Server-Codes erstellt als erstes ein neues Sysdeo-Tomcat Projekt. Auch hier ist die Nutzung des Generator-Konzeptes von Vorteil, da bei

```java
String[] natures = new String[]{JavaCore.NATURE_ID,};
IProject project = ResourcesPlugin.getWorkspace().getRoot().getProject(name);
IProjectDescription description = project.getDescription();
description.setNatureIds(natures);
project.setDescription(description, null);
IJavaProject javaProject = JavaCore.create(project);
```

Neben besonderen Naturen (zum Beispiel *com.sysdeo.eclipse.tomcat.tomcatnature*) müssen für das Tomcat Projekt noch verschiedene Classpath-Einstellungen und Bibliothekszuordnungen getroffen werden. So benötigt es beispielsweise einen Verweis auf die Bibliotheken des Tomcat-Containers und die Java Runtime. All diese Operationen wurden in einer eigenen Komponente gekapselt, damit die Erstellung von Projekten einfacher erfolgen konnte und bestimmte Operationen wieder verwendet werden können.


### 5.9.2 Generierung der Client-Erweiterung

Der Client-Generator erstellt, wie der Generator des Servers, ein Java-Projekt. Allerdings werden hier die spezifischen Naturen für *Fragmente* hinzugefügt. Neben diesen Informationen wird in die fragment.xml der Name des Host-Plugins, also des GMF-Diagram Plugins, generiert. Der Name dieses Plugins wird über das GMF-GenModel ermittelt und so die Namenskonvention des GMF-Projektes eingehalten. Alle Klassen der Editor-Erweiterung werden über einen *oAW-Workflow* erzeugt. Das entsprechende Template für den Quellcode erstellt dabei nicht nur die Package-Struktur, sondern passt auch generisch die Paket- und Klassenbezeichnungen an die des Hostprojektes an [vgl. Abbildung 65].

![Abbildung 65 - generiertes Client Projekt](image)

Mit die wichtigste Generierung betrifft hierbei den *ElementTypeHelper*, der die Zuordnung zwischen View und ElementType ermöglicht. Dabei greift das Template auf alle Knoten und Kanten im GMF-GenModel zu, welches die Information über die
Implementierung  158

Visual-ID eines jeden Knotens und dessen ElementType beinhaltet. Listing 19 zeigt einen Ausschnitt aus dem Template für die Generierung des ElementTypeHelpers. Der Übersicht halber sind Import-Statements, Kommentare und andere Deklarationen entfernt worden.

```
<DEFINE elementTypeHelperImpl FOR GenEditorGenerator>
<FILE>
"src/"+getDiagramPackagePath()+"/dawn/synchronization/impl/"+model
ID+"DawnElementTypeHelperImpl.java"
<ENDFILE>
<ENDDEFINE>

public class "modelID"DawnElementTypeHelperImpl implements
DawnElementTypeHelper {

public IElementType getElementType(View view) {

int visualId = "modelID"VisualIDRegistry.getVisualID(view);
switch (visualId) {

«EXPAND elementMapping(this.diagram) FOR this.diagram»
«FOREACH this.diagram.childNodes AS e»

«EXPAND elementMapping(this.diagram) FOR e»
«ENDFOREACH»

«FOREACH this.diagram.topLevelNodes AS e»
«EXPAND elementMapping(this.diagram) FOR e»
«ENDFOREACH»

«FOREACH this.diagram.links AS e»
«EXPAND elementMapping(this.diagram) FOR e»
«ENDFOREACH»

} 

return null;

<ENDDEFINE>

<DEFINE elementMapping(GenDiagram d) FOR GenDiagram>
  case «this.editPartClassName».VISUAL_ID:

  return d.elementTypesClassName."getUniqueIdentifierName(this.elementTyp
e.uniqueIdentifier)";

<ENDDEFINE>
```

Listing 19 - Ausschnitt oAW-Template für ElementTypeHelper

5.9.3 Generierung des Web-Viewers

Das Aussehen der Knoten und Kanten eines GMF-Editors kann vom GMF-Entwickler innerhalb des GMF-Graph Modells verändert werden [vgl. 2.8.7.1]. Er nutzt dabei einen baumbasierten Editor, um einzelne Elemente feingranularer zu gestalten. Zum Beispiel um zu einem Knoten Compartments und Unterknoten hinzuzufügen oder Kanten mit


Auf Grund der Fülle der Möglichkeiten einen GMF-Editor zu gestalten, wurden nicht alle möglichen Varianten umgesetzt. Die in Dawn aufgegriffenen Figurenbeschreibungen genügen aber, um einfache Editoren für das Web zu erstellen. Durch die einfache Handhabung des XPand-Template-Mechanismus können aber
zusätzliche Funktionen später einfach erstellt werden. Die wichtigsten Änderungen, auf die Dawn Einfluss nehmen kann, werden im Folgenden erläutert. Eine visualisierte Form des Generierungsprozesses findet sich unter [Flügge 2009b].

5.9.3.1 Knoten

Zur Erstellung rechteckiger Knoten kann innerhalb von OpenJacob Draw2D die Klasse `CompartmentFigure` genutzt werden. Diese erbt von der JavaScript-Klasse `Node` und kann zusätzlich Kindsobjekte verwalten, welche über entsprechende Methoden dem Objekt hinzugefügt werden können. Dies ermöglicht das dynamische Hinzufügen von Unterobjekten, wie das `Diagramm-Servlet` es nutzt. Da es aufwändig wäre diese Erweiterungen in jede zu erstellende Figure zu generieren, wurde eine Oberklasse für alle rechteckigen Knoten erstellt, von der die Figuren erben (`org.mftech.dawn.basic.DawnNodeFigure`). Dadurch kann das Generat klein und übersichtlich gehalten werden. Listing 20 stellt eine generierte Figure für einen rechteckigen Knoten dar.

```
1  org.mftech.diagram.uml.clazz.AClassFigure = function(className)
2  {
3      org.mftech.dawn.basic.DawnNodeFigure.call(this);
4      this.outputPort1 = null;
5      this.outputPort2 = null;
6      this.setDimension(50, 50);
7      this.setResizeable(false);
8      this.setClassName(className);
9  }
10 org.mftech.diagram.uml.clazz.AClassFigure.prototype = new
11 org.mftech.dawn.basic.DawnNodeFigure;
12
13 org.mftech.diagram.uml.clazz.AClassFigure.prototype.type =
14 "org.mftech.diagram.uml.clazz.AClassFigure";
15 org.mftech.diagram.uml.clazz.AClassFigure.prototype.
16 createHTMLElement = function()
17  {
18      var item = org.mftech.dawn.basic.DawnNodeFigure.prototype.
19         createHTMLElement.call(this);
20      item.style.border = "1px solid #B0B0B0";
21      item.style.backgroundColor = "rgb(230,230,255)";
22      return item;
23  }
Listing 20 - Generat - AClassFigure
```

GMF ist aber nicht nur auf rechteckige Knoten beschränkt. Über den Figure-Deskriptor `RoundedRectangle` können Knoten erzeugt werden, bei denen die Kanten abgerundet sind. Dabei kann GMF über die Parameter `CornerHeight` und `CornerWidth` den Grad

Um Knoten mit runden Ecken zu erzeugen, wurde ebenfalls eine Basisklasse erzeugt, welche einen runden Knoten darstellen kann. Das XPand-Template prüft, ob der Knoten normal oder rundkantig ist, indem er die Klasse des Modells ausliest. Danach entscheidet es, von welcher Klasse der Knoten erben soll. Ist die Basisklasse des Figure-Deskriptors RoundedRectangle, so parametrisiert er zusätzlich noch die Klassen mit dem Wert für die Kantenbiegung.

5.9.3.2 Kanten

Dawn stellt für generierte Kanten die Basis-Klasse DawnConnectionFigure zur Verfügung. Kantendefinitionen bestehen aus zwei Teilen, der eigentlichen Kante und Dekoratoren, welche das Aussehen der Kantenenden definieren. Hierbei besteht das Problem, dass Dekoratoren mit Hilfe von Template Points definiert werden können, um ihnen ein spezielles Aussehen zu verleihen [vgl. 2.8.7.3]. Leider stimmt die Koordinatenbeschreibung innerhalb von GMF nicht mit den Zeichen-Methoden der Draw2D API von OpenJacob überein. Infolgedessen können die Koordinaten nicht einfach übernommen werden. Um die Dekoratoren trotzdem anhand der Beschreibung im GMF-Graph-Modell für den Web-Viewer generieren zu können, wurde empirisch eine Transformationsfunktion ermittelt, mit der eine Transformation der TemplatePoints von dem einen in das andere Koordinatensystem erfolgen kann [vgl. Abbildung 67].

43 http://www.css3.info/border-radius-apple-vs-mozilla/
Zusätzlich zu dem Aussehen der Dekoratoren kann Dawn zusätzlich sowohl Einfluss auf die Kantenstärke als auch die Hintergrundfarbe im Fall einer *PolyGon Decoration* nehmen. Der Generator erkennt anhand des Modells welche Dekoratoren zu welchen Kanten zugeordnet sind und setzt die entsprechenden Beziehungen. Alle JavaScript-Dekoratoren erben von der Basisklasse *DawnBasicConnectionDecorator*. 

\[
    f(x, y) = \begin{pmatrix} -5x \\ 3y \end{pmatrix}
\]

*Abbildung 67 - Transformationsfunktion für Template Points*
6 Tests

6.1 Funktionale Testfälle


6.2 Komponentenorientierte Tests


\(^{44}\) http://www.junit.org/
6.3 Performance Tests


6.3.1 Testumgebung

Für die Performance Tests wurde eine Testumgebung bestehend aus einem Server und zwei Clients aufgebaut. Hierbei wurde ein Client mit aktueller Hardware und einer mit älterer Hardware genutzt, um auch schwächere Clients zu simulieren. Dabei wurde besonders bei der Übertragung mit SOAP ein deutlich erkennbarer Leistungseinbruch auf der älteren Hardware erwartet. Die software- und hardwaretechnische Ausstattung der Clients ist Tabelle 5 zu entnehmen. Server und Client wurden über einen 10/100 MBit Switch verbunden. Auf beiden Rechnern wurde Eclipse 3.4.0 (Ganymede Release) verwendet.

<table>
<thead>
<tr>
<th>System</th>
<th>Client 1</th>
<th>Client 2</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>AMD Duron (Morgan) 1,0 GHz</td>
<td>Intel Core2Duo (T7250); 2,0 GHz</td>
<td>Intel Core2Duo (E6600); 2,4 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>768 MB SD-RAM</td>
<td>2 GB/ DDR2</td>
<td>3,6 GB / DDR2</td>
</tr>
<tr>
<td>Festplatte</td>
<td>120 GB</td>
<td>180 GB (RAID 0)</td>
<td>300 GB (RAID 0)</td>
</tr>
<tr>
<td>Netzwerkarte</td>
<td>10/100 MBit</td>
<td>100/1000 MBit</td>
<td>100/1000 MBit</td>
</tr>
<tr>
<td>Betriebssystem</td>
<td>Windows 2000 SP4</td>
<td>Windows XP SP 3</td>
<td>Windows XP SP 3</td>
</tr>
</tbody>
</table>

Tabelle 5 - Systemumgebung für Performance Tests

Alle Untersuchungen liefen mit dem unmodifizierten Server. Für die Ausführung des Clients wurde ein kleines Testframework integriert, welches in einer gestarteten

6.3.2 Protokoll Vergleich

Die Vergleichstests basierten darauf, dass für beide Protokolle dieselben Abfolgen von Funktionen ausgeführt wurden. Um die Übertragungsgeschwindigkeit der Ressourcen zu testen, wurde ein spezielles Diagramm mit 35 Interfaces, 70 Klassen und 105 Verbindungen entworfen. Die ausgeführten Operationen für die Protokolle sind in Tabelle 6 aufgeleistet.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Erklärung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>Eine komplette Ressource wird an den Client übertragen, bestehend aus 70 Klassen, 35 Interfaces, 105 Konnektoren</td>
</tr>
<tr>
<td>Lock</td>
<td>5 Objekte werden gesperrt</td>
</tr>
<tr>
<td>unlock</td>
<td>5 Objekte werden entsperrt</td>
</tr>
<tr>
<td>User rights</td>
<td>Die Rechte des Nutzers werden abgefragt</td>
</tr>
<tr>
<td>getProjectNames</td>
<td>Alle Projektnamen werden vom Server geladen</td>
</tr>
<tr>
<td>getMyProjectNames</td>
<td>Alle Projektnamen des Nutzers werden vom Server geladen</td>
</tr>
</tbody>
</table>

Tabelle 6 - Ausgeführte Vergleichsoperationen

Zur Glättung der Testergebnisse wurden die Tests jeweils 1000-mal ausgeführt und der Durchschnittswert der einzelnen Ergebnisse ermittelt. Die Ausführungszeiten der einzelnen Operationen wurden in Millisekunden erfasst. Um die Ergebnisse nicht zu verfälschen, wurden die Tests auf beiden Clients zeitlich unabhängig voneinander ausgeführt.

Das ermittelte Ergebnis entspricht den vorher erwarteten Werten. Durch die Serialisierung der Methodenparameter und der Rückgabewerte nach XML produziert SOAP einen auffallenden Overhead gegenüber RMI. Besonders deutlich ist beim Vergleich zwischen Client 1 und Client 2 [vgl. Abbildung 68 und Abbildung 69] zu sehen, dass das Marshelling und Unmarshelling, also das Umwandeln der Transportdaten nach Java, bei SOAP einen starken Performanceeinbruch bewirkt, während sich RMI annähernd konstant verhält.
In Betrachtung dieser Ergebnisse empfiehlt sich bei Clients mit reduzierter Hardwareausstattung auf RMI zu wechseln. Allerdings muss in diesem Fall berücksichtigt werden, dass mögliche Firewall-Restriktionen die Kommunikation unterbinden können.

Die Abfragen der einzelnen Clients hatten auf den Server keinerlei Auswirkungen. Die Prozessorlast lag bei diesen Tests bei durchschnittlich 9 Prozent.
6.3.3 Skalierungstests


Abbildung 70 - Gesamtzeit mit steigender Client Anzahl

7 Zusammenfassung


7.1 Ergebnisbetrachtung


Die Verwendung von bereits in GMF-Editoren genutzten Ressourcen, als Transportmedium für die Übermittlung der Modelldaten, vereinfachte die Entwicklung hinsichtlich der Serialisierung der Daten. Diese wird direkt vom GMF-Framework
übernommen und musste nicht zusätzlich implementiert werden. Dadurch liegen die Daten in exakt derselben Form, wie auf dem Client vor. Da GMF nach XMI serialisiert, ist zusätzlich ein interoperables Austauschformat verfügbar, welches in späteren Entwicklungen auch von anderen Produkten genutzt werden kann.


Durch die Nutzung von modellgetriebenen Techniken zur Bereitstellung der editorspezifischen Erweiterungen integriert sich das Produkt in den Entwicklungszyklus von GMF-Editoren. Entwickler müssen also nicht umdenken und können einfach die Erweiterungen generieren. Dadurch, dass Dawn keine Änderungen am Editor vornimmt, können auch bereits existierende Editoren erweitert werden.

Allerdings kann in dem begrenzten Zeiträumen nicht jedes Problem für ein komplexes, verteiltes System bis ins kleinste Detail analysiert und behoben werden. Dies war auch nicht der Anspruch an diese Arbeit, sondern vielmehr das Aufzeigen von verschiedensten Möglichkeiten, welche zu weiteren Entwicklungen anregen sollen. Ziel


### 7.2 Schlussfolgerungen und Erfahrungen

Neben der Implementierung des Systems war es Ziel der Arbeit einen Einblick in die Welt der Eclipse-Entwicklung zu bekommen. Wenn auch anfangs das Bewusstsein vorhanden war, dass Eclipse mehr als eine Entwicklungsumgebung ist, so wurden die Erwartungen bei Weitem übertroffen. Die zahlreichen Eclipse-Projekte decken einen umfangreichen Bereich von softwaretechnischen Lösungen ab. Dabei ist nicht nur die Anwendung der Frameworks auf Nutzerebene interessant, sondern ebenfalls die dahinter liegende Softwarearchitektur.

Allerdings ist es im Eclipse-Umfeld besonders bei neueren Projekten oft schwer gezielte Informationen zu erhalten. So ist es besonders im Bereich von GMF kaum möglich auf zusammenhängende Printmedien zurückzugreifen. Dies zwingt zur Nutzung von nicht immer aktuellen Internet-Quellen, was die Einarbeitung in die Thematik schwieriger als erwartet gestaltete. Die dabei besten Quellen für Probleme und Lösungen sind die zahlreichen Eclipse-Newsgroups, bei denen schnell Lösungsansätze für bestehende Probleme bereitgestellt werden.

7.3 Zukünftige Erweiterungen


Neben diesen Erweiterungen gibt es eine Vielzahl kleiner Änderungen, die die Arbeit mit Dawn erleichtern könnten, sei es ein Button, mit dem die Verbindung zum Server hergestellt oder auf Wunsch unterbrochen werden kann, oder andere nützliche Hilfsmittel. Da Dawn als Open-Source Projekt weitergeführt wird, werden auch solche kleineren Ergänzungen über kurz oder lang implementiert werden.
Literaturverzeichnis

Burghardt 2004 Markus Burghardt:
Web-Services
Deutsche Universitäts-Verlag, Wiesbaden, 2004

Daum 2006 Dr. Berthold Daum:
Das Eclipse-Codebuch
dpunkt.verlag GmbH, Heidelberg, 2006

Daum 2007 Dr. Berthold Daum:
Rich-Client Entwicklung mit Eclipse 3.2 (2. Auflage)
dpunkt.verlag GmbH, Heidelberg, 2007

Daum 2007a Dr. Berthold Daum:
Java 6 programmieren mit der Java Standard Edition
Pearson Education, München, 2007

Funke 2009 Holfer Funke:
Devide et Impera: Das OSGi Framework
Eclipse Magazin 01/09, S.12-15
Software & Support Verlag, Frankfurt am Main, 2009

Gronback 2009 Richards Gronback:
Eclipse Modeling Project – A Domain-Specific Language (DSL) Toolkit
Pearson Education, Inc., Boston, 2009

Gruhn et al. 2006 Volker Gruhn, Daniel Pieper, Carsten Röttgers:
MDA®
Springer-Verlag, Berlin Heidelberg, 2006

Haiges 2006 Sven Haiges:
Neues von der Tag-und-Nachtgleiche
Eclipse Magazin 03/06, S.35-37
Software & Support Verlag, Frankfurt am Main, 2006
Haischt 2006  
Daniel S. Haischt:  
**Das Medium**  
Eclipse Magazin 03/06, S.76-78  
Software & Support Verlag, Frankfurt am Main, 2006

Hennig et al 2008  
Manfred Hennig, Markus Seeberger:  
**Einführung in den „Extension-Point“ Mechanismus von Eclipse**  
JavaSpektrum 01/2008, S.19-24  
SIGS DATACOM GmbH, Troisdorf, 2008

Henning 2003  
Dr. Peter A. Henning:  
**Taschenbuch Multimedia**  
Carl Hanser Verlag, München Wien, 2003

Jung 2002  
Volker Jung, H.-J. Warnecke:  
**Handbuch für die Telekommunikation (2. Auflage)**  
Alex Springer Verlag, Berlin, 2002

Künneth 2008  
Thomas Künneth:  
**Einstieg in Eclipse 3.4 (2. aktualisierte und erweiterte Auflage)**  
Galileo Press, Bonn, 2008

Melzer et al. 2008  
Ingo Melzer et al.:  
**Service-orientierte Architekturen mit Web-Services (3. Auflage)**  
Spektrum Akademischer Verlag, Heidelberg, 2008

Merks et al. 2009  
Ed Merks, Davis Steinberg, Frank Budinsky, Marcelo Peternostro:  
**Eclipse Modelling Framework (2nd Edition)**  
Addison-Wesley Longman, Amsterdam, 2009

Oates et al. 2008  
Richard Oates, Thomas Langer, Stefan Wille, Torsten Lueckow, Gerald Bachlmayer:  
**Spring & Hibernate: Eine praxisbezogene Einführung (2. Auflage)**  
Carl Hanser Verlag, München Wien, 2008
<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Publisher/Place</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>John Wiley &amp; Sons Ltd., Sussex/UK</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Carl Hanser Verlag, München Wien</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Galileo Press, Bonn</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>O'Reilly Verlag GmbH &amp; Co. KG, Köln</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JavaSpektrum 01/2006; S.14-17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SIGS DATACOM GmbH, Troisdorf</td>
<td></td>
</tr>
<tr>
<td>Schumacher 2008</td>
<td>Edwin Schumacher:</td>
<td><em>Teamwork ohne Kompatibilitätsgrenzen</em></td>
<td>2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse Magazin 04/08, S.75-78</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Software &amp; Support Verlag, Frankfurt am Main</td>
<td></td>
</tr>
<tr>
<td>Schuster 2008</td>
<td>Andreas Schuster:</td>
<td><em>Komplett Effizient</em></td>
<td>2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse Magazin 01/08, S.43-48</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Software &amp; Support Verlag, Frankfurt am Main</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java Magazin 1/2009; S.26-32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Software &amp; Support Verlag, Frankfurt am Main</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Software &amp; Support Verlag GmbH, München</td>
<td></td>
</tr>
</tbody>
</table>
Stahl et al. 2007  Thomas Stahl, Markus Völter, Sven Efftinge, Arno Haase:
Modellgetriebene Softwareentwicklung (2. Auflage)
dpunkt.verlag GmbH, Heidelberg, 2007

Stepper 2009  Eike Stepper:
Modelling goes Enterprise
Eclipse Magazin 03/09, S.38-42
Software & Support Verlag, Frankfurt am Main, 2009

Szypersky et al. 2002  Clemens Szypersky, Dominik Gruntz, Stephan Murer:
Component Software, Beyond Object Oriented Programming (2nd Edition)
Person Education, New York, 2002

Tanenbaum 2001  Andrew S. Tanenbaum:

Tanenbaum 2003  Andrew S. Tanenbaum:
Computer Networks (International Edition)
Pearson Education Inc., New Jersey, 2003

Tanenbaum et al. 2003  Andrew S. Tanenbaum, Maarten van Steen:
Verteilte Systeme – Grundlagen und Paradigmen
Pearson Education Deutschland GmbH, München, 2003

Teufel 2008  Mark Teufel:
Odysee im Weltraum?
Eclipse Magazin 04/08, S.13-16
Software & Support Verlag, Frankfurt am Main, 2008

Wenz 2008  Christina Wenz:
JavaScript und Ajax (8. Auflage)
Galileo Press, Bonn, 2008

Widder 2004  Oliver Widder:
Adapter in Eclipse
Eclipse Magazin 01/04, S.70-80
Software & Support Verlag, Frankfurt am Main, 2004
Wunderlich 2006  Lars Wunderlich:
    **Calisto in Mind**
    Eclipse Magazin 04/06, S.21-23
    Software & Support Verlag, Frankfurt am Main, 2006

Wütherich et al. 2008  Gerd Wütherich, Nils Hartmann, Bernd Kolb, Matthias Lübken:
    **Die OSGi Service Platform**
    dpunkt.verlag GmbH, Heidelberg, 2008

Zeitner et al. 2008  Alfred Zeitner, Birgit Linner, Martin Maier:
    **Spring 2.5: Eine pragmatische Einführung**
    Pearson Education, München, 2008
Internetquellen

Alle Internetquellen sind in dem zuletzt geprüften Zustand auf der beigefügten CD hinterlegt.

ComputerZeitung
2009
Susanne Franke:
**OSGi-Framework bezieht weitere Java-Rahmenwerke in die Spezifikation ein**
http://www.computerzeitung.de/articles/osgi-framework_bezieht_weitere_java-rahmenwerke_in_die_spezifikation_ein:/2009010/31856360_ha_CZ.html?thes=9845,10228,10231&tp=/ausrichtungen/prozesse
*zuletzt geprüft: 08.05.2009*

ECF 2008
ECF:
**Extending Real-Time Shared Editing for Use with Other Editors**
http://wiki.eclipse.org/Extending_Real-Time_Shared_Editing_for_Use_with_Other_Editors
*zuletzt geprüft: 08.05.2009*

Eclipse 2009
Eclipse Community:
**GEF Programmer’s Guide (Release 3.4 Ganymede)**
*zuletzt geprüft: 08.05.2009*

Eclipse 2009a
Eclipse Foundation:
**Eclipse Projects Overview**
*zuletzt geprüft: 08.05.2009*

EMFT 2008
Laurent Goubet:
**[EMF Compare] Problem comparing identical GMF-resources**
http://dev.eclipse.org/newslists/news.eclipse.technology.emf/msg06068.html
*zuletzt geprüft: 08.05.2009*
<table>
<thead>
<tr>
<th>Internetquellen</th>
<th>Martin Flügge:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flügge 2009a</strong></td>
<td><strong>Screencast – Generating the source</strong>&lt;br&gt;<a href="http://www.mftech.org/dawn/screencasts/12_generating%20the%20source/generating.htm">http://www.mftech.org/dawn/screencasts/12_generating%20the%20source/generating.htm</a>&lt;br&gt;<em>zuletzt geprüft: 08.05.2009</em></td>
</tr>
<tr>
<td><strong>Flügge 2009b</strong></td>
<td><strong>Screencast – Generating JavaScript Figures</strong>&lt;br&gt;<a href="http://www.mftech.org/dawn/screencasts/10_generationJavascript/10_generationJavascript.htm">http://www.mftech.org/dawn/screencasts/10_generationJavascript/10_generationJavascript.htm</a>&lt;br&gt;<em>zuletzt geprüft: 08.05.2009</em></td>
</tr>
<tr>
<td><strong>Flügge 2009c</strong></td>
<td><strong>Screencast – Locking</strong>&lt;br&gt;<a href="http://www.mftech.org/dawn/screencasts/9_locking/locking.htm">http://www.mftech.org/dawn/screencasts/9_locking/locking.htm</a>&lt;br&gt;<em>zuletzt geprüft: 08.05.2009</em></td>
</tr>
<tr>
<td><strong>Flügge 2009d</strong></td>
<td><strong>Screencast – Conflicts</strong>&lt;br&gt;<a href="http://www.mftech.org/dawn/screencasts/5_Conflicts/5_Conflicts.htm">http://www.mftech.org/dawn/screencasts/5_Conflicts/5_Conflicts.htm</a>&lt;br&gt;<em>zuletzt geprüft: 08.05.2009</em></td>
</tr>
<tr>
<td><strong>Flügge 2009e</strong></td>
<td><strong>Screencast – Creating Projects</strong>&lt;br&gt;<a href="http://www.mftech.org/dawn/screencasts/6_creating_projects/creating_projects_new.htm">http://www.mftech.org/dawn/screencasts/6_creating_projects/creating_projects_new.htm</a>&lt;br&gt;<em>zuletzt geprüft: 08.05.2009</em></td>
</tr>
<tr>
<td><strong>Flügge 2009f</strong></td>
<td><strong>Screencasts – More sophisticated diagrams</strong>&lt;br&gt;<a href="http://mftech.org/dawn/screencasts/8_more%20sophisticated%20Diagrams/8_more_sophisticated_Diagrams.htm">http://mftech.org/dawn/screencasts/8_more%20sophisticated%20Diagrams/8_more_sophisticated_Diagrams.htm</a>&lt;br&gt;<em>zuletzt geprüft: 08.05.2009</em></td>
</tr>
<tr>
<td><strong>GMF 2009</strong></td>
<td><strong>Eclipse Foundation:</strong>&lt;br&gt;<strong>GMFgraph Hints</strong>&lt;br&gt;<a href="http://wiki.eclipse.org/GMFGraph_Hints">http://wiki.eclipse.org/GMFGraph_Hints</a>&lt;br&gt;<em>zuletzt geprüft: 08.05.2009</em></td>
</tr>
<tr>
<td>Internetquellen</td>
<td>Titel</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| GMF 2009a      | Seweryn Niemiec, Jaap Retimes and Richard Gronback and others:  
GMF Tutorial – Part 2  
http://wiki.eclipse.org/GMF_Tutorial_Part_2 | | |
| GMF 2009b      | Eclipse Foundation:  
Developer Guide to the GMF Runtime Framework  
| IBM 2008       | Chris Aniszczyk:  
Plug-in development 101, Part 1: The fundamentals  
| Mammana 2008   | Jean-Charles Mammana, Romain Meson, Jonathan Gramain:  
GEF (Graphical Editing Framework) Tutorial  
| OMG 2006       | Object Management Group:  
MOF Core specification Version 2.0  
http://www.omg.org/docs/formal/06-01-01.pdf | | |
| OSGi 2009      | OSGi Alliance:  
Automotive Electronics Market  
http://www.osgi.org/Markets/Automotive | | |
| Raible 2007    | Matt Raible:  
Comparing Java Web Frameworks  
<table>
<thead>
<tr>
<th>Jahr</th>
<th>Autor</th>
<th>Thema</th>
<th>URL</th>
<th>Letztes geprüft: 08.05.2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUN 2008</td>
<td>SUN:</td>
<td>GlassFish V3 runs on OSGi</td>
<td><a href="http://blogs.sun.com/dochez/entry/glassfish_v3_runs_on_osgi">http://blogs.sun.com/dochez/entry/glassfish_v3_runs_on_osgi</a></td>
<td></td>
</tr>
<tr>
<td>Abbildungsverzeichnis</td>
<td>Seite</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 1 - Beziehung System-Modell-Metamodell</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 2 - Beispiel abstrakte/konkrete Syntax</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 3 - Beispiel Stadtplan</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 4 - Transformation von PIM bis zum Quellcode</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 5 - MOF Architekturmodell</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 6 - Architektur der OSGi-Frameworks</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 7 - einfacher OSGi-Management Agent</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 8 - Bundle Lebenszyklus</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 9 - Architektur von Eclipse</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 10 - PDE User Interface</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 11 - Plugin Mechanismus</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 12 - Nutzungs- und Abhängigkeitsrichtung bei Vererbung</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 13 - Nutzungs- und Abhängigkeitsrichtung bei Extension Points</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 14 – vereinfachtes EMF Ecore Modell</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 15 - Darstellung von MVC in GEF</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 16 - Ausführen eines Kommandos</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 17 - Einfacher mit GMF erzeugter Editor</td>
<td>46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 18 - GMF-Modelle</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 19 - GMF-Architektur</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 20 - Webframeworks - Vergleich Jobbörsen</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 21 - Webframeworks - Vergleich Xing</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 22 - Webframeworks - Vergleich Literatur</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 23 - Anwendungsfall: Überblick</td>
<td>62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 24 - vereinfachte Systemdarstellung</td>
<td>63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 25 - Überblick über die einzelnen Sub-Projekte</td>
<td>71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 26 - Verteilungsdiagramm des Systems</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 27 - Projekte</td>
<td>74</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 28 - System Design</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 29 - Publish/Update</td>
<td>79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 30 - Full-Sync-Delete Anomalie</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 31 - Lokal- und Remote-Änderungskonflikt</td>
<td>87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 32 - Lokaler Löschkonflikt</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 33 - Entfernter Löschkonflikt</td>
<td>89</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 34 - Änderungskonflikte serverseitig</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 35 - Änderungskonflikt serverseitig - Lösung</td>
<td>91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 36 - Algorithmus zur Konflikterkennung</td>
<td>94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 37 - Algorithmus zur Konfliktbehebung lokal geänderter Objekte</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 38 - Flussdiagramm: Locking im Offline-Modus</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 39 – Klassendiagramm: Adapter mit Offline-Klassen</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbildung 40 - Rechtesystem in Dawn</td>
<td>103</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abbildungsverzeichnis

Abbildung 41 - Unterschiedliche Referenzierung von UserManagern ......................... 104
Abbildung 42 - ERM Diagramm des Persistenzmodells ............................................ 106
Abbildung 43 - Web-Viewer Architektur .................................................................. 111
Abbildung 44 - Sitemap - Web-GUI ....................................................................... 113
Abbildung 45 - Architektur des Clients .................................................................. 114
Abbildung 46 - prototypisches Klassendiagramm .................................................... 116
Abbildung 47 - Dawn Prototyp Ecore-Modell ......................................................... 117
Abbildung 48 - Dawn-GenModel Ecore................................................................... 119
Abbildung 49 - Dawn Information ......................................................................... 128
Abbildung 50 - Wizards zum Erstellen und Beitreten von Projekten ....................... 130
Abbildung 51 - Anlegen der Action für die Konfliktbehandlung............................... 131
Abbildung 52 - Editor mit Konflikten ..................................................................... 132
Abbildung 53 - Dawn Conflict View ....................................................................... 133
Abbildung 54 - Locken eines Objektes ................................................................... 135
Abbildung 55 - Extension Point Schema Editor....................................................... 136
Abbildung 56 - Dawn Preferences .......................................................................... 138
Abbildung 57 - DBManager ..................................................................................... 141
Abbildung 58 - Zuordnung GMF-Modell zu JavaScript-Views ................................... 144
Abbildung 59 - GMF- und JavaScript-Knoten im Vergleich ...................................... 146
Abbildung 60 - Web-Viewer (links) und GMF-Editor (rechts)................................... 148
Abbildung 61 - Web-UI ............................................................................................ 150
Abbildung 62 - Kontext-Menü auf einer Projekt Datei ............................................. 151
Abbildung 63 - Web-UI innerhalb von Eclipse .......................................................... 152
Abbildung 64 - Menü auf dem Dawn-GenModel....................................................... 155
Abbildung 65 - generiertes Client Projekt ................................................................. 157
Abbildung 66 - Generierung Web-Viewer ................................................................. 159
Abbildung 67 - Transformationsfunktion für Template Points .................................. 162
Abbildung 68 - Vergleich SOAP/RMI – Client 1....................................................... 166
Abbildung 69 - Vergleich SOAP/RMI – Client 2....................................................... 166
Abbildung 70 - Gesamtzeit mit steigender Client Anzahl .......................................... 167
## Tabellenverzeichnis

<table>
<thead>
<tr>
<th>Tabelle</th>
<th>Titel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabelle 1</td>
<td>Die 13 Diagramme der UML 2</td>
<td>19</td>
</tr>
<tr>
<td>Tabelle 2</td>
<td>Open-Source OSGi-Implementierungen</td>
<td>27</td>
</tr>
<tr>
<td>Tabelle 3</td>
<td>Überblick über die Eclipse Projekte im Ganymede Release</td>
<td>31</td>
</tr>
<tr>
<td>Tabelle 4</td>
<td>Preferences</td>
<td>115</td>
</tr>
<tr>
<td>Tabelle 5</td>
<td>Systemumgebung für Performance Tests</td>
<td>164</td>
</tr>
<tr>
<td>Tabelle 6</td>
<td>Ausgeführte Vergleichsoperationen</td>
<td>165</td>
</tr>
</tbody>
</table>

## Listings

<table>
<thead>
<tr>
<th>Listing</th>
<th>Titel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Listing 1</td>
<td>OSGi-Manifest</td>
<td>25</td>
</tr>
<tr>
<td>Listing 2</td>
<td>Dynamic EMF</td>
<td>42</td>
</tr>
<tr>
<td>Listing 3</td>
<td>Mapping von ElementType zur JavaScript-View</td>
<td>110</td>
</tr>
<tr>
<td>Listing 4</td>
<td>Zeitverzögertes Starten der Dawn Komponenten</td>
<td>124</td>
</tr>
<tr>
<td>Listing 5</td>
<td>Auswahl des Kommunikationsprotokolls</td>
<td>125</td>
</tr>
<tr>
<td>Listing 6</td>
<td>editorspezifisches ResourceSet</td>
<td>126</td>
</tr>
<tr>
<td>Listing 7</td>
<td>getElementType(View)</td>
<td>127</td>
</tr>
<tr>
<td>Listing 8</td>
<td>DawnDiagramEditorInterface</td>
<td>129</td>
</tr>
<tr>
<td>Listing 9</td>
<td>Dawn Extension Service</td>
<td>137</td>
</tr>
<tr>
<td>Listing 10</td>
<td>Zugriff auf das Workspace-Root</td>
<td>140</td>
</tr>
<tr>
<td>Listing 11</td>
<td>HibernateCreationUtil</td>
<td>141</td>
</tr>
<tr>
<td>Listing 12</td>
<td>config.xml Schema</td>
<td>143</td>
</tr>
<tr>
<td>Listing 13</td>
<td>vereinfachtes Modell (semantic und notational)</td>
<td>145</td>
</tr>
<tr>
<td>Listing 14</td>
<td>Ausschnitt - erzeugter Diagram Quelltext</td>
<td>147</td>
</tr>
<tr>
<td>Listing 15</td>
<td>exemplarische config.xml</td>
<td>147</td>
</tr>
<tr>
<td>Listing 16</td>
<td>Web-Viewer JSP - Ausschnitt</td>
<td>149</td>
</tr>
<tr>
<td>Listing 17</td>
<td>Beispiel-Anfrage mittels SSO</td>
<td>153</td>
</tr>
<tr>
<td>Listing 18</td>
<td>Erstellung eines Projektes in Eclipse</td>
<td>156</td>
</tr>
<tr>
<td>Listing 19</td>
<td>Ausschnitt oAW-Template für ElementTypeHelper</td>
<td>158</td>
</tr>
<tr>
<td>Listing 20</td>
<td>Generat - AClassFigure</td>
<td>160</td>
</tr>
<tr>
<td>Listing 21</td>
<td>einfacher Unit-Test</td>
<td>164</td>
</tr>
</tbody>
</table>
Glossar


API  Application Programming Interface bezeichnet eine Schnittstelle für die Entwicklung von Anwendungen, die von einem System bereitgestellt wird.


Axis  Ist ein Framework für die Erstellung von SOAP-Anwendungen der Apache Software Foundation.

CDO  Connected Distributed Objects ist ein Projekt der Eclipse Foundation, welches sich mit der Verteilung von EMF-Modellen beschäftigt.

CIM  Das Computation Independent Model ist im Sinne der MDSD ein Modell, welches unabhängig von jeglicher Implementierung ist. Es beschreibt das System ohne Hinblick auf die spätere Umsetzung.

CMOF  Complete Meta Object Facility bezeichnet den vollständigen Umfang der Meta Object Facility Spezifikation der OMG.


CSS  Cascading Style Sheets ist eine Stylesheet-Sprache für HTML- und XML-basierte Dokumente. Mit Hilfe von CSS kann die Formatierung dieser Dokumente beeinflusst werden.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domäne</td>
<td>Domänen innerhalb der MDSD bezeichnen ein in sich geschlossenes Anwendungsgebiet. Sie bilden die Grundlage für die Entwicklung von domänenspezifischen Sprachen.</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language (domänenspezifische Sprachen) bezeichnet eine Sprache mit Bezug zu einer Domäne innerhalb der MDSD. DSLs können sowohl textuell, als auch grafisch sein.</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Bezeichnet sowohl eine integrierte Entwicklungsumgebung, als auch ein Open-Source Projekt.</td>
</tr>
<tr>
<td>ECOOP</td>
<td>European Conference of Object-Oriented Programming. Eine jährlich stattfindende Konferenz auf der Themen rund um die Objektorientierte Programmierung diskutiert werden.</td>
</tr>
<tr>
<td>EMOF</td>
<td>Das Essential Meta Object Facility bildet den essentiellen Kern der MOF-Spezifikation der OMG wieder.</td>
</tr>
<tr>
<td>Equinox</td>
<td>Bezeichnet die OSGi-Implementierung von Eclipse und den Kern von RCP-Anwendungen.</td>
</tr>
<tr>
<td>GEF</td>
<td>Graphical Editing Framework beschäftigt sich mit der Bereitstellung von Werkzeugen und Architekturen zur einfachen Entwicklung von grafischen Editoren basierend auf Eclipse.</td>
</tr>
<tr>
<td>Glossar</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Hibernate</td>
<td>Ist ein objektrelationaler Mapper zum Umwandeln von objektorientierten Sachverhalten auf eine relationale Datenbank.</td>
</tr>
<tr>
<td>HQL</td>
<td>Hibernate Query Language bezeichnet eine an SQL angelehnte Abfragesprache innerhalb von Hibernate. Im Gegensatz zu SQL werden allerdings Objekte und nicht Relationen abgefragt.</td>
</tr>
<tr>
<td>IRC</td>
<td>Internet Relay Chat ist ein textbasiertes System zum Austausch von Chat-Nachrichten zwischen mehreren Nutzern.</td>
</tr>
<tr>
<td>Java EE / JEE</td>
<td>Java Enterprise Edition ist eine Plattform von Sun zur Entwicklung von Unternehmensanwendungen.</td>
</tr>
<tr>
<td>Java RE / JRE</td>
<td>Java Runtime Environment bezeichnet die Laufzeitumgebung der Programmiersprache Java. Sie umfasst unter anderem die Virtuelle Maschine von Java.</td>
</tr>
</tbody>
</table>
JET | Java Emitter Templates bezeichnet eine Template-basierte Sprache zur Transformation von Modellen in andere Modelle oder Texte.

JFace | JFace ist ein UI-Bibliothek, welche auf SWT aufsetzt und das Programmieren SWT-basierter Oberflächen wesentlich vereinfacht.

JMS | Java Messaging Service ist ein von SUN spezifizierter Standard zur nachrichtenbasierten Übertragung von Informationen.


LAN | Local Area Network bezeichnet ein lokales Netzwerk, welches in der Regel auf TCP/IP beruht. LANs sind in der Regel räumlich begrenzt und erstrecken sich selten über mehr als ein Gebäude.


MDA | Model Driven Architecture. Ein Ansatz zur Entwicklung von modellgetriebener Software. Basiert auf dem MOF der OMG.

MDD | Model Driven Development. Siehe MDSD.

MDSD | Model Driven Software Development. Ein konzeptueller Ansatz zur Entwicklung von modellgetriebener Software.

MDSE | Model Driven Software Engineering. Siehe MDSD.

MOF | Meta Object Facility bezeichnet das Meta-Modell der OMG, welchem auch die UML zugrunde liegt.
MVC  Model View Control ist ein Entwurfsmuster zur Trennung von Anzeige und Datenhaltung.

oAW  openArchitectureWare ist ein Framework, welches modellgetriebene Entwicklungsprozesse unterstützt.


OOSE  Object-Oriented Software Engineering. Eine Entwurfsmethode für objektorientierte Software.

ORM  Object Relational Mapping bezeichnet das Abbilden von Objekten auf relationale Datenbanken.

OSGi  Open Services Gateway Initiative ist eine Organisation, welche die OSGi Services Platform spezifiziert. Zuweilen wird auch diese Plattform, beziehungsweise die dahinter stehenden Technologien, als OSGi bezeichnet.

PDE  Plugin Development Environment ist ein Eclipse-Projekt zur Entwicklung von Plugins und OSGi-Bundles.


Plattform  Plattformen bezeichnen im Kontext der MDSD Systeme, die als Grundlage für die Generierung genutzt werden. Plattformen können zum Beispiel Betriebssysteme oder Programmiersprachen sein.

PSM  Platform Specific Model bezeichnet ein Modell innerhalb der MDSD welches einen Bezug zu einer Plattform hat.

QVT  Query View Transformation ist eine Spezifikation der OMG innerhalb des MOF zur Umsetzung von Modell-zu-Modell-Transformationen.

<table>
<thead>
<tr>
<th>Glossar</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST</td>
<td>Representational State Transfer bezeichnet ein HTTP-basiertes Kommunikationsmodell. Es wird häufig für die Implementierung von Webservices genutzt.</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation bezeichnet den Aufruf einer Methode eines Objektes innerhalb eines verteilten Systems. Ebenfalls wird unter RMI die entsprechende Implementierung für Java bezeichnet.</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call. Ein entfernter Prozedur-Aufruf. Im Gegensatz zur RMI findet dieser nicht an einem Objekt statt.</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture bezeichnet ein Konzept zur Nutzung von Diensten in einer meist webbasierten Architektur.</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language ist eine Abfragesprache für relationale Datenbanken.</td>
</tr>
<tr>
<td>SWT</td>
<td>Standard Widget Toolkit. Eine UI-Bibliothek von IBM. Die Eclipse IDE basiert auf SWT.</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunication System bezeichnet das Mobilfunksystem der dritten Generation. UMTS ist der Nachfolger von GMS.</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier beschreibt ein universelles Format zur Beschreibung von Ressource Locations.</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless LAN bezeichnet einen Standard zur kabellosen Übertragung von Daten.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language ist eine Sprache zur plattformunabhängigen Beschreibung von Webservice-Schnittstellen.</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Meta Interchange ist ein interoperables XML-basiertes Austauschformat der OMG für MOF-konforme Modelle.</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language. Eine Auszeichnungssprache für strukturierte Inhalte. XML ist textbasiert und die Basis für viele verschiedene Anwendungen.</td>
</tr>
</tbody>
</table>
Index

A

Abstract Window Toolkit .................................................. 42
AJAX ..................... Siehe Asynchronous JavaScript and XML 38
Ansicht ........................................................................ 52
Authentifizierung ......................................................... 101
Autorisierung ............................................................... 101
AWT ..................... Siehe Abstract Window Toolkit 52
Axis ........................................................................... 71

B

Bundles ...................................................................... 24

C

CDO ............. Siehe Connected Distributed Objects 41
CIM .................. Siehe Computation Independent Model 34
Computation Independent Model ...................................... 13
Computed Distributed Objects ...................................... 137
ContentProvider .......................................................... 74

D

Datenübertragung .......................................................... 79
Dawn ................................................................. 71, 121
DawnRemoteConnection ................................................... 73, 124
DawnRemoteConnector .................................................... 76, 124
DOM ..................... Siehe Document Object Model 52
DOM .......................... Siehe Document Object Model 52
Document Object Model ............................................... 52
Domain Specific Language Siehe domänen-spezifische Sprache 12
domänen-spezifische Sprache .......................................... 12
Draw2D ................................................................. 43
DSL ..................... Siehe domänen-spezifische Sprache 43
Dynamic EMF ............................................................. 41

E

ECF .......... Siehe Eclipse Communication Framework 40
Editor ................................................................. 38
Eclipse ................................................................. 30
EMF .................................................. Siehe Eclipse Modeling Framework 40
Codegen .............................................................. 40
Core ................................................................. 39
dynamic ............................................................. 39
Edit ................................................................. 40
Equinox .............................................................. 32
Equinox .............................................................. 32
Extension ............................................................. 35
Extension Point ....................................................... 36

F

Figure Desriptor .......................................................... 49
Fragment ................................................................. 38

G

GEF ..................... Siehe Graphical Editing Framework 17
Generativer Programmierung ........................................ 17
Generator ............................................................... 15
GlobalUserManager ................................................... 104
GMF .................. XXII, Siehe Graphical Modelling Framework 12
Graphical Modelling Framework .................................. 43

H

Hibernate ................................................................. 52, 73, 105

J

Java Emitter Templates ................................................. 57
Java Server Faces ..................................................... 58
JET ..................... Siehe Java Emitter Templates 58
JFace ................................................................. 44
JSF ............. Siehe Java Server Faces 43
JUnit ..................... Siehe Java Server Faces 163

K

Konfliktbeseitigung ...................................................... 94
Konflikte ............................................................... 85
Beseitigung ....................................................... Siehe Konfliktbeseitigung 94

L

LaB:Sis ................................................................. 158
LaB:Sis ................................................................. 158
Lohner ................................................................. 158
Lohner ................................................................. 158

N

Navigator ................................................................. 158
Navigator ................................................................. 158

O

Observer ................................................................. 158
Observer ................................................................. 158

P

Perspektive ............................................................. 38
Perspektive ............................................................. 38
Plug-in ................................................................. 40
Platform ............................................................... 38

R

Ressource .............................................................. 75

S

Server ................................................................. 71, 155

T

Text ................................................................. 71
Web-Viewer ........................................................... 142, 158
Wizards ................................................................. 115, 130

W

Wizards ................................................................. 115, 130
Wizards ................................................................. 115, 130
Wizards ................................................................. 115, 130
Wizards ................................................................. 115, 130
Wizards ................................................................. 115, 130
Index

Entfernter Löschkonflikt .......................................................... 88
Erkennung ..................Siehe Konflikt-erkennung durch einen Entfernten Löscherkennung ...88
Lokal-und Remote-Änderungskonflikt .......................... 86
Lokaler Löschkonflikt .......................................................... 87
serverseitig ................................................................. 89
Vermeidung .......Siehe Konfliktvermeidung durch einen Serverseitigen Löschervermeidung ...93
Konfliktvermeidung ................................................................ 97, 134

L

LocalSession............................................................. 153

M

M2M ..................Siehe 3.5.3.4 Modell-Transformation
M2T ..................Siehe Model-To-Text Transformation
MDA ..................Siehe Model Driven Architecture
MDSD .................. 11, 15
Meta Object Facility ..................................................... 20
Meta-modell .................. 10
Meta-modell .................. 8
Model Driven Architecture .................................................. 17
Modell .................. 6
formales ................................................................. 7
Modell-Transformation .................................................. 14
Model-to-Code Transformation Siehe Model-To-Text Transformation
Model-To-Text Transformation .................................................. 12
MOF ..................Siehe Meta Object Facility
MyFaces .................. 58

N

Netzwerkanabhängigkeit ................................................ 98
NullUserManager ..................... 104

O

oAW ..................Siehe openArchitectureWare
openArchitectureWare .................................................. 57
Workflow .................................................. 57
XPand ..................................................Siehe XPand
OSGi ..................Siehe 3.7 OSGi Service Platform
 Bundle .................................................. Siehe Bundle Framework
Implementierungen .................................................. 22
Service .................................................. Siehe Service
OSGi Service Platform .................................................. 21

P

PDE ............ Siehe Plugin Development Environment
Persistenz .................. 105
PIM ............ Siehe Platform Independent Model
Platform Independent Model .................................................. 13
Platform Specific Model .................................................. 14
Plattform .................................................. 13
Plugin .................................................. 84
Plugin Development Environment .................................................. 33
PSM ..................Siehe Platform Specific Model

Publish ................................................................. 79

R

RCP ..................Siehe Rich Client Platform
RemoteConnection ....Siehe DawnRemoteConnection
RemoteConnector ......Siehe DawnRemoteConnector
Representational State Transfer .................................................. 64
Request ............................................................. 44
Resource ............................................................. 41
ResourceSet ............................................................. 41
REST ...........Siehe Representational State Transfer
Rich Client Platform .................................................. 38
Rich Client Plattform .................................................. 33

S

Serviceorientierte Architektur .................................................. 28
Services .................................................. 26
Shale .................................................. 58
Simple Object Access Protocol ............................................. 51
SOA ...........Siehe Serviceorientierte Architektur
SOAP ...........Siehe Simple Object Access Protocol
Spring MVC ............................................. 58
Standard Widget Toolkit .................................................. 43
StandardUserManager .................................................. 103
Stripes .................................................. 58
Struts .................................................. 58
Swing .................................................. 42
SWT ...........Siehe Standard Widget Toolkit
Synchronisation .................................................. 78
Syntax .................................................. 9
abstrakt .................................................. 9
konkret .................................................. 9

T

Tapestry ................................................................. 58

U

UML ............ 8, Siehe Unified Modeling Language
Unified Modeling Language ............................................. 18
Update .................................................. 79
UserManager .................................................. 102

W

WDSL ...........Siehe Web Service Description Language
Web Framework .................................................. 58
Web Service .................................................. 51
Web Service Description Language ............................................. 51
Web-Viewer .................................................. 107

X

X MI ................................................................. 41
XML-RCP .................. 51
XPand .................................................. 57
Anhang

A Anforderungskatalog

A.1 Einleitung

Der nachfolgende Anforderungskatalog bildet die Grundlage für die Entwicklung einer kollaborativen, webbasierten Erweiterung für GMF-Editoren, nachfolgend als Dawn bezeichnet.

A.2 Zielbestimmungen


A.2.1 Musskriterien

Kommunikation

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-KM-01</td>
<td>Firewall Transparenz</td>
<td>Das System soll unabhängig von Firewall-Konfigurationen arbeiten können.</td>
</tr>
<tr>
<td>M-KM-02</td>
<td>Ausfallsicherheit und Transparenz</td>
<td>Das System soll sich robust gegen Ausfall des Servers verhalten.</td>
</tr>
<tr>
<td>M-KM-03</td>
<td>Netzwerk-unabhängigkeit</td>
<td>Der Nutzer soll unabhängig von einer Netzwerkverbindung am Editor arbeiten können. Bei bestehender Verbindung sollen die Modellbestände synchronisiert werden.</td>
</tr>
<tr>
<td>M-KM-04</td>
<td>Starting Watcher-Thread when opening editor</td>
<td>Wenn der Editor gestartet wird, soll, wenn der Server vorhanden ist, automatisch eine Verbindung aufgenommen werden.</td>
</tr>
<tr>
<td>M-KM-05</td>
<td>Stopping Watcher-Thread when closing editor</td>
<td>Wenn der Editor geschlossen wird, soll die Verbindung sauber abgebaut werden.</td>
</tr>
<tr>
<td>M-KM-06</td>
<td>Open Exchange Format</td>
<td>Die Kommunikation soll über ein interoperables Austauschformat erfolgen, um später andere Clients (ggf. andere Programmiersprachen) mit in das System einbinden zu können.</td>
</tr>
</tbody>
</table>
### Ressourcenmanagement und Synchronisation

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-RS-01</td>
<td>Synchronize Changed Nodes - Client</td>
<td>Geänderte Knoten sollen auf dem Client mit dem Server synchronisiert werden können. Die GUI soll die Änderungen anzeigen.</td>
</tr>
<tr>
<td>M-RS-02</td>
<td>Synchronize Changed Nodes - Server</td>
<td>Geänderte Knoten sollen auf dem Server mit den Client Informationen synchronisiert werden können.</td>
</tr>
<tr>
<td>M-RS-03</td>
<td>Synchronize Inserted Nodes - Client</td>
<td>Eingefügte Knoten sollen auf dem Client mit dem Server synchronisiert werden können. Die GUI soll die Änderungen anzeigen.</td>
</tr>
<tr>
<td>M-RS-04</td>
<td>Synchronize Inserted Nodes - Server</td>
<td>Eingefügte Knoten sollen auf dem Server mit den Client Informationen synchronisiert werden können.</td>
</tr>
<tr>
<td>M-RS-06</td>
<td>Synchronize Deleted Nodes - Server</td>
<td>Gelöschte Knoten sollen auf dem Server mit den Client Informationen synchronisiert werden können.</td>
</tr>
<tr>
<td>M-RS-08</td>
<td>Synchronize Changed Edges - Server</td>
<td>Geänderte Kanten sollen auf dem Server mit den Client Informationen synchronisiert werden können.</td>
</tr>
<tr>
<td>M-RS-09</td>
<td>Synchronize Inserted Edges - Client</td>
<td>Eingefügte Kanten sollen auf dem Client mit dem Server synchronisiert werden können. Die GUI soll die Änderungen anzeigen.</td>
</tr>
<tr>
<td>M-RS-10</td>
<td>Synchronize Inserted Edges - Server</td>
<td>Eingefügte Kanten sollen auf dem Server mit den Client Informationen synchronisiert werden können.</td>
</tr>
<tr>
<td>M-RS-13</td>
<td>Object-Identification via XMI-Ids</td>
<td>Alle EObjects, auch die Elemente in den Views, sollen über eine XMI ID identifiziert werden können.</td>
</tr>
<tr>
<td>M-RS-14</td>
<td>Store Diagram in DB</td>
<td>Der aktuelle Zustand des Diagramms soll auf dem Server in einer Datenbank gespeichert werden.</td>
</tr>
<tr>
<td>M-RS-17</td>
<td>Conflict-Update-Blocking</td>
<td>Wenn Konflikte erkannt werden, dann soll keine Information zum Server übertragen werden. Das Übertragen der Informationen an den Server soll so lange unterbunden werden, bis alle Konflikte beseitigt wurden.</td>
</tr>
<tr>
<td>M-RS-18</td>
<td>Conflict Detection</td>
<td>Konflikte sollen im System erkannt werden können.</td>
</tr>
<tr>
<td>M-RS-19</td>
<td>Conflict Avoidance</td>
<td>Konflikte sollen vom System wenn möglich vermieden werden.</td>
</tr>
<tr>
<td>M-RS-20</td>
<td>Conflict Resolving</td>
<td>Konflikte sollen im System gelöscht werden können.</td>
</tr>
<tr>
<td>M-RS-21</td>
<td>Change multiple objects before publish</td>
<td>Bevor der Modellbestand an den Server geschickt werden soll, soll es möglich sein, dass mehrere Objekte geändert werden können. Dies bedingt, dass lokale Änderungen erkannt und geblockt werden müssen.</td>
</tr>
<tr>
<td>M-RS-22</td>
<td>Joined resource files</td>
<td>Die Synchronisierung soll für alle GMF Resource funktionieren, und denen GMF und EMF Resource in einer Datei gespeichert werden</td>
</tr>
<tr>
<td>M-RS-23</td>
<td>Optimistic Locking</td>
<td>Durch ein optimistische Lock-Verfahren soll verhindert werden, dass Race-Conditions auftreten können</td>
</tr>
<tr>
<td>M-RS-24</td>
<td>Pessimistic Locking</td>
<td>Durch ein manuelles pessimistisches Lockverfahren sollen Konflikte vermieden werden.</td>
</tr>
</tbody>
</table>
**User Management**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-UM-01</td>
<td>Add User to Project</td>
<td>Benutzer sollen Projekten hinzugefügt werden können.</td>
</tr>
<tr>
<td>M-UM-02</td>
<td>Store User-Information in DB</td>
<td>Die Nutzerinformationen sollen in der Datenbank gespeichert werden.</td>
</tr>
<tr>
<td>M-UM-03</td>
<td>Verschlüsselte Passwörter</td>
<td>Passwörter der Nutzer sollen verschlüsselt abgelegt werden.</td>
</tr>
<tr>
<td>M-UM-04</td>
<td>Join only to assigned projects</td>
<td>User dürfen nur Projekte joinen, zu denen sie eingeladen wurden.</td>
</tr>
</tbody>
</table>

**Eclipse-Editor**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-EE-01</td>
<td>User-Login</td>
<td>Benutzer sollen sich über die Eclipse GUI am System anmelden können.</td>
</tr>
<tr>
<td>M-EE-02</td>
<td>User-Logoff</td>
<td>Nutzer sollen sich über die Eclipse GUI vom System abmelden können.</td>
</tr>
<tr>
<td>M-EE-03</td>
<td>Preferences – Server</td>
<td>Der Nutzer soll über die Editor Preferences die Angaben zum Server editieren können. Hierzu zählen URL und Port-Nummer</td>
</tr>
<tr>
<td>M-EE-04</td>
<td>Preferences – Update Intervals</td>
<td>Der Nutzer soll über die Eclipse Preferences die Angabe der Update Intervals bestimmen können. Die Angebe erfolgt in Millisekunden</td>
</tr>
<tr>
<td>M-EE-05</td>
<td>Create Projects</td>
<td>Über den Eclipse Client sollen Projekte auf dem Server angelegt werden können.</td>
</tr>
<tr>
<td>M-EE-06</td>
<td>Visuals for Conflict Detection</td>
<td>Das System soll eine geeignete Visualisierung für Konflikte innerhalb des Editors anzeigen</td>
</tr>
<tr>
<td>M-EE-07</td>
<td>GUI Components for Conflict Resolving</td>
<td>Es sollen dem Nutzer Dialoge zur Verfügung gestellt werden, um Konflikte zu beseitigen</td>
</tr>
<tr>
<td>M-EE-08</td>
<td>Project-Management Dialog</td>
<td>Es soll einen Dialog geben, über den der Initialtor eines Project die Nutzer verwalten kann</td>
</tr>
</tbody>
</table>

**Web-Komponenten**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-WE-01</td>
<td>User-Login</td>
<td>Benutzer sollen sich über das Web-Frontend am System anmelden können.</td>
</tr>
<tr>
<td>M-WE-02</td>
<td>User-Logoff</td>
<td>Nutzer sollen sich über das Web-Frontend vom System abmelden können.</td>
</tr>
<tr>
<td>M-WE-03</td>
<td>Web-Viewer</td>
<td>Über das Web-Frontend sollen alle Änderungen der Eclipse Nutzer nachvollzogen werden können</td>
</tr>
</tbody>
</table>

**Generierung**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-GE-01</td>
<td>Generation Model Wizard</td>
<td>Über einen Wizard soll das PSM für die Netzwerkschicht erstellt werden können</td>
</tr>
<tr>
<td>M-GE-02</td>
<td>Generation Option</td>
<td>Über eine geeignete Option soll der Quellcode für den Server und das Client-Erweiterungsplugin erzeugt werden.</td>
</tr>
<tr>
<td>M-GE-03</td>
<td>Generation for simple Editors</td>
<td>Die Generierung der Netzwerkschicht soll für einfache GMF-Editoren möglich sein.</td>
</tr>
<tr>
<td>M-GE-04</td>
<td>Generate GMF-Figure for Web</td>
<td>Es die Repräsentation der GMF-Figures für den Web-Viewer generiert werden.</td>
</tr>
</tbody>
</table>
A.2.2 Wunschkriterien

**Kommunikation**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
</table>
| W-KM-01 | Dynamic Network Communication       | Das System soll selbstständig entscheiden können, wie es die Daten verschickt. Zwei Möglichkeiten stehen zur Verfügung: a) die Daten werden als Plain XML verschickt  
  - Vorteil: Client und Server sparen Rechenzeit  
  - Nachteil: Langsame Verbindung bei großen Projekten  
  b) die Daten werden gezipped verschickt  
  - Vorteil: schnellere Verbindung bei großen Projekten  
  - Nachteil: mehr Rechenarbeit bei Server und Client |
| W-KM-02 | Extensible Network Layer            | Das zugrunde liegende Netzwerkprotokoll soll geändert, bzw. erweitert werden können. Soll es möglich sein, dass System später auf andere Technologien wie zum Beispiel RMI, JMS, aufbauen zu können. |
| W-KM-03 | Runtime System Update               | Das System soll in der Lage sein durch ein Update zur Laufzeit mit neuen Editoren umgehen zu können. |
| W-KM-04 | Chat                                | Die Nutzer sollen sich über einen Chat unterhalten können. |
| W-KM-05 | Audio-Communication                 | Die Nutzer sollen sich über eine Sprachverbindung (z.B. Skype) unterhalten können. |
| W-KM-06 | Information via email               | Nutzer sollen per Mail über die Einladung in ein Project informiert werden können. |
| W-KM-07 | Secure Communication               | Die Netzwerkverbindung soll verschlüsselt erfolgen können. |

**Ressourcenmanagement und Synchronisation**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>W-RS-01</td>
<td>Multi-Object-Editing</td>
<td>Mehrere Nutzer sollen gleichzeitig an einem Objekt arbeiten können, solange sie nicht am selben Bereich des Objektes arbeiten.</td>
</tr>
<tr>
<td>W-RS-02</td>
<td>Multiple resource files</td>
<td>Die Synchronisierung soll auch für alle GMF Projekte Funktionieren, und denen GMF und EMF Ressource in unterschiedlichen Datei gespeichert werden.</td>
</tr>
<tr>
<td>W-RS-03</td>
<td>Store Diagram History in DB</td>
<td>Die History des Diagramms soll in der Datenbank gespeichert werden.</td>
</tr>
</tbody>
</table>

**Eclipse-Editor**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
</table>
| W-EE-01 | Change User Data Extended           | Der Dialog für die Nutzerdaten soll um folgende Eigenschaften erweitert werden  
  - ICQ Nummer  
  - Skype Nummer  
  - Nickname |
| W-EE-02 | Internationalisierung               | Die Dialoge sollen über Fragmentprojekte internationalisierbar sein. |
### Anhang XXXIV

| W-EE-03 | Preference Publish-Mode | Über den Publish Mode soll bestimmt werden, wie publiziert wird. „on save“ oder „on interval“ |
| W-EE-04 | Locking not available while offline | Das Locking Menü darf nicht angezeigt werden, wenn der Server nicht verfügbar ist. |
| W-EE-09 | Change User Data | Es soll einen Dialog geben, über den der Benutzer seine Daten ändern kann: -Name -Passwort -email |

### Web-Editor

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>W-WE-01</td>
<td>Create Projects</td>
<td>Über das Web-Frontend sollen Projekte auf dem Server angelegt werden können.</td>
</tr>
<tr>
<td>W-WE-02</td>
<td>Edit Diagram</td>
<td>Das Diagram soll über das Web-Frontend bearbeitet werden können.</td>
</tr>
<tr>
<td>W-WE-03</td>
<td>Project-Management Dialog (Web-Based)</td>
<td>Es soll einen Dialog geben, über den der Initiator eines Projekts die Nutzer verwalten kann</td>
</tr>
<tr>
<td>W-WE-04</td>
<td>Change User Data (Web-based)</td>
<td>Es soll einen Dialog geben, über den der Benutzer seine Daten ändern kann: -Name -Passwort - Nickname</td>
</tr>
</tbody>
</table>

### Generierung

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>W-GE-01</td>
<td>Generation for complex GMF-Editors</td>
<td>Die Generierung der Netzwerkschicht soll für komplexe GMF-Editoren möglich sein.</td>
</tr>
</tbody>
</table>

### A.2.3 Abgrenzungskriterien

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-AK-01</td>
<td></td>
<td>Die Generierung basiert ausschließlich auf GMF, andere Generierungen werden nicht in Betracht gezogen.</td>
</tr>
<tr>
<td>A-AK-03</td>
<td></td>
<td>Die Implementierung erfolgt ausschließlich in Java &gt;= 1.5. Andere Programmiersprachen werden nicht berücksichtigt.</td>
</tr>
<tr>
<td>A-AK-05</td>
<td></td>
<td>Das Testen der Funktionalität erfolgt ausschließlich unter Windows XP. Unter anderen Betriebssystemen finden keine Tests statt.</td>
</tr>
<tr>
<td>A-AK-06</td>
<td></td>
<td>Das System wird ausschließlich mit den unter 0 aufgeführten Systemen und Komponenten entwickelt und getestet. Entwicklungen und Tests mit bzw. unter anderen Komponenten finden nicht statt.</td>
</tr>
</tbody>
</table>
**A.3 Produkteinsatz**


**A.4 Produktumgebung**

**A.4.1 Software**

<table>
<thead>
<tr>
<th>Softwarekomponente</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.5</td>
</tr>
<tr>
<td>GMF</td>
<td>2.1.0</td>
</tr>
<tr>
<td>EMF</td>
<td>2.4.0</td>
</tr>
<tr>
<td>Eclipse</td>
<td>3.4.0 Ganymede</td>
</tr>
<tr>
<td>MySQL</td>
<td>3.21</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>5.x</td>
</tr>
<tr>
<td>Apache Axis</td>
<td>1.3</td>
</tr>
</tbody>
</table>

**A.4.2 Hardware**

<table>
<thead>
<tr>
<th>Softwarekomponente</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client für kollaborativen GMF-Editor</td>
<td>Internetfähiger Rechner, der den Anforderungen an das aktuelle Eclipse Release genügt.</td>
</tr>
<tr>
<td>Cient für Web-Editor</td>
<td>Internetfähiger Rechner, der über einen Mozilla-kompatiblen Browser verfügbar.</td>
</tr>
</tbody>
</table>
| Server                        | Internetfähiger Rechner, der den Anforderungen an das aktuelle Eclipse Release genügt. Genügend Festplattenplatz, die die Speicherung der Daten der Projekte.
B Inhalt der beigefügten CD

- CD
  - Quellcode
    - DawnServer
    - org.mftech.dawn.codegen
    - org.mftech.dawn.codegen.edit
    - org.mftech.dawn.codegen.editor
    - org.mftech.dawn.runtime.client
    - org.mftech.diagram.uml.class
    - org.mftech.diagram.uml.class.diagram
    - org.mftech.diagram.uml.class.diagram.communication
    - org.mftech.diagram.uml.class.diagram.extension
    - org.mftech.diagram.uml.class.edit
    - org.mftech.diagram.uml.class.editor
  - Weblinks
    - ComputerZeitung 2009
    - ECF 2008
    - Eclipse 2009
    - Eclipse 2009a
    - EMFT 2008
    - Flügge 2009a
    - Flügge 2009b
    - Flügge 2009c
    - Flügge 2009d
    - Flügge 2009e
    - GMF 2009
    - GMF 2009a
    - IBM 2008
    - Mammana 2008
    - OSGi 2009
    - Schnepel 2008
    - Stepper 2008
    - SUN 2008
    - Voelter Blog 2007
    - GMF 2009b
    - Raible 2007
    - OMG 2006