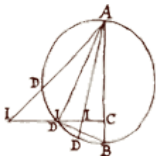
Diplomarbeit

Entwicklung eines MDD-Tools für eine virtuelle Ausstellung

Julia Damerow

Technische Fachhochschule Berlin
University of Applied Sciences



Max-Planck-Institut für
Wissenschaftsgeschichte

Eingereicht am Fachbereich VI Informatik und Medien
der Technischen Fachhochschule Berlin
Studiengang Medieninformatik Diplom
Schwerpunkt Software
Matrikelnummer 730187

Betreuer Prof. Dr. Sebastian von Klinski
Gutachterin Prof. Dr. Karin Schiele

Bearbeitungszeitraum: 05.05.2008 - 08.08.2008

Die vorliegende Diplomarbeit mit dem Titel “Entwicklung eines MDD-Tools für eine virtuelle Ausstellung” wurde selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst. Alle Stellen der Arbeit, die anderen Werken wörtlich oder sinngemäß entnommen sind, sind unter Angabe der Quelle kenntlich gemacht Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Julia Damerow
4. August 2008

«Die Wirklichkeit ist öfter ungenau.»

(Douglas Adams, aus: Das Restaurant am Ende des Universums)

Inhaltsverzeichnis

Kapitel 1.

Einleitung

Die vorliegende Diplomarbeit wurde am Max-Planck-Institut für Wissenschaftsgeschichte (MPIWG) angefertigt. Ziel der Arbeit ist es zu entscheiden, ob der Ansatz der modellgetriebenen Softwareentwicklung für die Neuentwicklung eines Systems zur Umsetzung von virtuellen Ausstellungen geeignet ist. Hierfür wird ein Prototyp, der diesem Ansatz folgt, konzipiert und umgesetzt.

Bei der modellgetriebenen Softwareentwicklung sind Modelle das zentrale Konzept. Der Begriff des "Modells" wird jedoch nicht ausschließlich nur in der Softwaretechnik verwendet. In vielen wissenschaftlichen Disziplinen hingegen existiert der Modellbegriff. Allgemein formuliert ist ein Modell hierbei eine *«idealisierte, vereinfachte, in gewisser Hinsicht ähnliche Darstellung eines Gegenstandes, Systems oder sonstigen Weltausschnitts* [...]*»* ([Sch99], S. 132 und [Hes94]). Ein Modell wird dabei als Nachbild der Realität aufgefasst, das der Erklärung dient ([Sch99]). Beispielsweise werden in der Mathematik und Physik Modelle zur Erklärung der Realität herangezogen. Im Gegensatz dazu werden Modelle in der Softwaretechnik als Beschreibung aufgefasst. Ein Modell dient hierbei der Spezifikation eines Systems. Im Rahmen der Softwareentwicklung wird somit unter einem Modell nicht ein Nachbild sondern das Vorbild eines Systems verstanden. Die Problematik dieser unterschiedlichen Auffassungen des Modellbegriffs beschreibt Schefe ausführlich in seinem Artikel "Softwaretechnik und Erkenntnistheorie" ([Sch99]).

Mittels des im Rahmen dieser Diplomarbeit entwickelten Prototyps werden Modelle von virtuellen Ausstellungen erzeugt. Eine virtuelle Ausstellung wird durch eine Website repräsentiert. Die Modelle dienen dabei als Vorbild für diese Websites. Sie sind zum Teil jedoch auch gleichzeitig das Nachbild eines realen Systems. Sie können zum Beispiel das Nachbild einer real existierenden Ausstellung sein.

1.1. Aufbau dieser Diplomarbeit

Im folgenden Kapitel wird zunächst die Ausgangssituation, die Anlass für diese Diplomarbeit war, erläutert. Im Rahmen dessen wird die Entstehungsgeschichte des Konzepts von virtuellen

Ausstellungen beschrieben. In Kapitel 3 werden dann die unterschiedlichen Techniken, die zur Umsetzung des Prototyps angewandt werden können, miteinander verglichen. Das Ergebnis des Vergleichs ist die Auswahl der modellgetriebenen Softwareentwicklung. Kapitel 4 führt anschließend die Begriffe und Konzepte der modellgetriebenen Softwareentwicklung ein und erläutert diese. Dieses Kapitel dient dem Verständnis der folgenden Teile dieser Arbeit. Im Anschluss daran werden in Kapitel 5 eine Reihe von unterschiedlichen Technologien vorgestellt, die zur Umsetzung des Ansatzes der modellgetriebenen Softwareentwicklung genutzt werden können.

Kapitel 6 beschreibt zunächst das System, das momentan zur Erstellung von virtuellen Ausstellungen genutzt wird. Im Anschluss daran werden in diesem Kapitel die Anforderungen, die an das zu entwickelnde System gestellt werden, definiert. Im Hinblick auf die seitens des MPIWG an das System gestellten Anforderungen und Bedingungen werden dann in Kapitel 7 die Techniken zur Umsetzung des Prototyps ausgewählt. Anschließend werden die Konzeption und die Implementierung des Prototyps beschrieben. Dabei wird eine Auswahl von Implementierungsdetails ausführlicher dargestellt.

Zur Bewertung des Prototyps wurden im Rahmen dieser Diplomarbeit unter anderem Anwendertests durchgeführt. Der Ablauf und die Resultate der Anwendertests werden in Kapitel 8 ausgeführt. Die Schlussfolgerungen, die aus der Entwicklung des Prototyps und aus den Anwendertests gezogen werden, sind in Kapitel 9 beschrieben. Schließlich wird in Kapitel 10 die weitere Entwicklung des Prototyps dargelegt und es werden Anregungen zur Überarbeitung des Prototyps gegeben. Darüber hinaus befinden sich in den Anhängen Ergänzungen zu den aufgeführten Kapiteln. Diese Ergänzungen bestehen hauptsächlich aus Screenshots, Bildern und Quelltexten.

1.2. Textauszeichnung

Zur besseren Lesbarkeit werden in dieser Diplomarbeit spezielle Begriffe innerhalb des Textes unterschiedlich ausgezeichnet. Im Text oder im Glossar erläuterte Begriffe werden dazu bei ihrem ersten Auftreten geneigt dargestellt: *erklärter Begriff*. Bei einem erneuten Vorkommen werden solche Begriffe nicht besonders ausgezeichnet. Zu diesen Begriffen werden hier auch die Projektbezeichnungen der vorgestellten Technologien und Begriffe, für die im darauffolgenden Text eine Abkürzung verwendet wird, gezählt.

Firmennamen werden dagegen bei jedem Vorkommen ausgezeichnet. Für die Auszeichnung werden Kapitälchen verwendet: Firmenname. Gibt es für einen Firmennamen eine Abkürzung, wird diese jedoch nicht speziell ausgezeichnet.

Begriffe aus Quelltextauszügen oder Diagrammen, wie beispielsweise Klassennamen, werden durch die Verwendung einer dicktengleichen Schriftart ausgezeichnet: `Klassenname`. Solche

Begriffe werden bei jedem Vorkommen ausgezeichnet. Für Quelltextauszüge (Listings) wird ebenfalls diese Schriftart verwendet. Zusätzlich sind diese grau hinterlegt.

Kapitel 2.

Ausgangssituation

Das MPIWG ist eines der 80 Forschungsinstitute der Max-Planck-Gesellschaft (MPG). Es beschäftigt sich mit der Frage *«unter welchen historischen Voraussetzungen wissenschaftliche Kultur und Wissenschaft als eine Kultur entstanden sind»* ([Max07]).[1] Im Rahmen der unterschiedlichen Projekte des Instituts wird dabei die Entwicklung und Anwendung neuer Softwaresysteme gefördert. Insbesondere das Internet wird zur Präsentation und Umsetzung einzelner Projekte genutzt.

Abb. 2-1. Medienstation in der Einsteinausstellung

Vom 16. Mai bis 30. September 2005 präsentierte die MPG im Berliner Kronprinzenpalais die vom MPIWG konzipierte Ausstellung "Albert Einstein — Ingenieur des Universums". Anlass hierfür war das Einsteinjahr 2005. Die Inhalte der Ausstellung umfassten zum einen die Person und das Leben Albert Einsteins. Zum anderen stellten sie seine wissenschaftlichen Leistungen in einen gesamtwissenschaftlichen, kulturellen und gesellschaftlichen Kontext. Die Ausstellung "Albert Einstein — Ingenieur des Universums" wird im Folgenden auch als *Einsteinausstellung* bezeichnet.

Parallel zur "realen Ausstellung" wurde eine "virtuelle Ausstellung", nachfolgend *virtuelle Einsteinausstellung* genannt, entwickelt. Die virtuelle Einsteinausstellung macht alle Ausstellungsräume und -inhalte über das Internet zugänglich. Hierfür wurden Fotos der Ausstellungsräume und Ausstellungsobjekte und die Texte zu den Ausstellungsobjekten auf einer Website veröffentlicht. Die virtuelle Einsteinausstellung ist auch nach Beendigung der Ausstellung im Kronprinzenpalais und damit auch zum Zeitpunkt der Anfertigung dieser Diplomarbeit weiterhin verfügbar.[2]

Wichtiger Bestandteil des Konzepts der Ausstellung "Albert Einstein — Ingenieur des Universums" waren die über alle Ausstellungsräume verteilten *Medienstationen*. Eine solche Medienstation ist in Abbildung 2-1 dargestellt. Technisch betrachtet war eine Medienstation ein

[1] Weitere Informationen zum MPIWG finden sich unter http://www.mpiwg-berlin.mpg.de/
[2] Die virtuelle Einsteinausstellung ist unter http://www.einsteinausstellung.de/ zu finden.

Touchscreen. Aus konzeptioneller Sicht erweiterte und vertiefte eine Medienstation die Inhalte eines Ausstellungsraumes. Sie stellte dafür den Besuchern der Ausstellung Informationen zu den Ausstellungsobjekten sowie allgemeines Hintergrundwissen zur Verfügung. Zum Teil wurden die Medienstationen dabei über Beamer an die Wand projiziert. Dadurch konnten sich mehrere Besucher gleichzeitig den Inhalt einer Medienstation ansehen.

Das aus der realen Einsteinausstellung übernommene Konzept der Medienstationen ist auch ein wesentlicher Bestandteil der virtuellen Einsteinausstellung. Neben den Fotos der Ausstellungsobjekte stellen die für die Medienstationen aufbereiteten und kommentierten Materialien die Hauptinhalte in der virtuellen Ausstellung dar. Die Medienstationen dienen dabei der Vermittlung und Veranschaulichung von Wissen und Informationen.

Das System zur Umsetzung von virtuellen Ausstellungen wurde am MPIWG mit finanzieller Unterstützung durch die Heinz Nixdorf Stiftung entwickelt. Es ermöglicht die internetgestützte Präsentation von Ausstellungsinhalten. Es wurde inzwischen von anderen Institutionen für eigene virtuelle Ausstellungen übernommen. Die Einsteinausstellung in Pavia[3] beispielsweise wurde ebenfalls mittels dieses Systems online zugänglich gemacht. Ein anderes Projekt, welches die Technologie der virtuellen Ausstellung nutzt, ist das virtuelle Museum des IES Canarias Cabrera Pinto[4].

[3] Die Einsteinausstellung in Pavia wurde ausgerichtet von der Universität von Pavia. Sie basiert auf den Entwicklungen im Rahmen der Berliner Einsteinausstellung. Die virtuelle Einsteinausstellung von Pavia ist unter http://einstein-pavia.mpiwg-berlin.mpg.de/ zu finden.

[4] Die virtuelle Ausstellung des virtuellen Museums des IES Canarias Cabrera Pinto ist unter http://fcohc.mpiwg-berlin.mpg.de:8080/CabreraPinto/ zu finden

Kapitel 3.

Umsetzungsstrategien

Das bisher genutzte System zur Umsetzung von virtuellen Ausstellungen am MPIWG, das *Altsystem*, wurde unter dem Namen *Virtual Exhibition Project* entwickelt. Bei der Benutzung des Systems fielen Defizite auf, so dass eine Neuentwicklung eines Systems zur Erstellung von virtuellen Ausstellungen als notwendig erachtet wurde. Im Rahmen dieser Diplomarbeit sind ein Konzept für das neue System und ein Prototyp entwickelt worden. Im Folgenden werden dafür grundlegende Umsetzungsstrategien vorgestellt und im Anschluss daran verglichen. Als Ergebnis des Vergleichs wird dabei eine Strategie zur Umsetzung des Prototyps ausgewählt.

3.1. HTML

Eine Strategie zur Umsetzung von virtuellen Ausstellungen ist die *Hypertext Markup Language* (HTML). HTML ist eine Beschreibungssprache für Dokumente. Sie dient dazu, Dokumente über das Internet zur Verfügung zu stellen. Ein mit HTML beschriebenes Dokument kann in einem Webbrowser dargestellt werden. HTML legt dabei die Struktur und die Darstellung eines Dokumentes fest. Das Dokument kann Text, Bilder und andere multimediale Inhalte enthalten. Des Weiteren ermöglicht HTML die Verwendung von *Hyperlinks*. Dies sind anklickbare Verweise, die unterschiedliche Dokumente miteinander verknüpften. ([Bar03], S. 431)

HTML wird unterschieden in *statisches HTML* und *dynamisches HTML*. Ursprünglich wurde HTML für die Bereitstellung von statischen Inhalten entwickelt. Ein Dokument, das mit statischem HTML beschrieben ist, ändert daher weder seinen Inhalt noch seine Darstellung, nachdem es im Webbrowser angezeigt wurde. Bei dynamischem HTML, sogenanntem *Dynamic HTML* (DHTML), kann sich im Gegensatz dazu ein Dokument auch dann noch ändern, wenn es bereits im Webbrowser angezeigt wird. DHTML ermöglicht dadurch die Interaktion zwischen Benutzer und Dokument. DHTML ist eine Kombination aus HTML, JavaScript und CSS. ([Nie02], Kap. 29)

Bei der Verwendung von HTML zur Erstellung einer virtuellen Ausstellung würden für jeden Raum, jede Medienstation und jedes Exponat HTML-Dokumente erstellt werden. Für die

Navigation durch die Website würden Hyperlinks erstellt. Die Verwendung von HTML für eine virtuelle Ausstellung hätte unterschiedliche Nachteile:

- Die Entwickler würden Kenntnisse in der Entwicklung von HTML-Dokumenten benötigen. Die Verwendung von DHTML würde darüber hinaus erfordern, dass die Entwickler Kenntnisse in JavaScript und CSS hätten.
- Die Wartung der Website wäre sehr umfangreich, da eine durchschnittliche virtuelle Ausstellung 100 und mehr Webseiten umfasst.
- Eine Übersicht über die gesamte virtuelle Ausstellung wäre nicht möglich. Ein Dokument gäbe jeweils nur Auskunft über Hyperlinks, die von diesem Dokument ausgehen. Es wäre nicht möglich zu erfahren, ob beziehungsweise wie viele Hyperlinks es gibt, die auf das Dokument verweisen, ohne alle anderen Webseiten der virtuellen Ausstellung zu untersuchen.

3.2. Web Application Framework

Heutzutage werden oft *Web Application Frameworks* für die Umsetzung von Webseiten eingesetzt. Ein Web Application Framework, auch als *Web-Framework* bezeichnet, ist ein Framework, das die Entwicklung von dynamischen Websites und Web-Anwendungen unterstützt und beschleunigt. Es stellt typischerweise häufig benötigte Funktionen durch wiederverwendbare Komponenten zur Verfügung. Diese Komponenten unterstützen beispielsweise die Navigation durch eine Website, das Logging oder den Zugriff auf eine Datenbank. Web-Frameworks sind beispielsweise *Ruby on Rails* oder *JavaServer Faces* (JSF). ([Ave06], Kap. 18.2)

Bei der Entwicklung von virtuellen Ausstellungen mit Hilfe von Web-Frameworks, würde der Entwickler ausgeprägte technische Kenntnisse benötigen. Die notwendigen Kenntnisse sind dabei abhängig von dem verwendeten Web-Framework. Für die Entwicklung mit Ruby on Rails beispielsweise werden Kenntnisse in der Programmiersprache *Ruby* benötigt. Für die Verwendung von JSF sind Kenntnisse in der Programmiersprache *Java* notwendig.

Für die Umsetzung von virtuellen Ausstellungen mittels eines Web-Frameworks könnten beispielsweise alle Inhalte der Ausstellung, wie die Texte zu den Exponaten oder die Beschreibungen der Räume, in einer Datenbank gespeichert werden. Entsprechend dem verwendeten Web-Framework würden dann Klassen oder Skripte erstellt, die die Anzeige der in der Datenbank gespeicherten Daten verwalten.

3.3. Content Management System

Die Verwaltung von Daten ist eine immer wiederkehrende Problemstellung bei der Entwicklung von Softwaresystemen. Speziell für die Erstellung und Verwaltung von Text- und

Multimedia-Dokumenten wurden daher *Content Management Systeme* (CMS) entwickelt. In der Regel dient ein CMS dazu, den Prozess der Erstellung, Verwaltung und Veröffentlichung von Inhalten zu kontrollieren. Im Rahmen dieser Diplomarbeit werden an dieser Stelle serverseitige Content Management Systeme betrachtet. Bei serverseitigen CMS werden die Inhalte in der Regel über eine Website veröffentlicht. Häufig wird für die Entwicklung eines serverseitigen CMS dabei ein Web-Framework verwendet. Ein CMS unterstützt die Verwaltung von Inhalten durch Funktionen wie zum Beispiel die Versionierung von Inhalten, die Verwaltung von Querverweisen und die Verwaltung von Benutzern. ([Nix05], Kap. 1)

Content Management Systeme folgen dem Prinzip, Inhalte und Präsentation zu trennen. Dadurch benötigt der Benutzer des CMS keine Kenntnisse über die technischen Details der Präsentation. Der Benutzer muss nur über das notwendige Fachwissen für die Erstellung und Pflege der Inhalte verfügen. ([Nix05], Kap. 1)

Bei der Entwicklung einer virtuellen Ausstellung mittels eines CMS wäre die Eingabe der fachlichen Daten unabhängig von der Darstellung der virtuellen Ausstellung. Die Informationen zu den Räumen der Ausstellung oder den Exponaten könnten ohne technisches Hintergrundwissen in dem CMS erfasst werden. Ein Entwickler mit den notwendigen technischen Kenntnissen würde das CMS konfigurieren und die benötigten Dateien für die Darstellung der virtuellen Ausstellung erstellen.

3.4. Modellgetriebene Softwareentwicklung

Web-Frameworks wie *Grails*[1] oder *Django*[2] unterstützen die Entwicklung von Web-Anwendungen unter anderem durch die automatische Erzeugung von Code. Bei diesen Web-Frameworks werden beispielsweise die *CRUD-Funktionen* für ein Objekt automatisch generiert. Die CRUD-Funktionen sind Funktionen zum Erstellen (Create), Abrufen (Retrieve), Ändern (Update) und Löschen (Delete) eines Objektes.

Diesem Konzept der automatischen Erzeugung von Code folgt auch die *modellgetriebene Softwareentwicklung*. Hierbei wird aus Modellen eines Softwaresystems automatisch der benötigte Quelltext erzeugt. Für virtuelle Ausstellungen würde das bedeuten, dass zunächst Modelle der Ausstellungen erstellt werden. Diese Modelle würden beispielsweise die Ausstellungsräume sowie die Exponate, die jeweils in den Räumen zu sehen sind, beschreiben. Anschließend würde aus diesen Beschreibungen der Ausstellungen der Quelltext für die Webseiten erzeugt werden.

Die modellgetriebene Softwareentwicklung ist ein Vorgehensmodell zur Softwareentwicklung, das in den vergangenen Jahren eine immer größerer Bedeutung gewonnen hat ([Sta07], Kap. 1.1). Der Einsatz von modellgetriebener Softwareentwicklung verspricht dabei, dass eine Reihe

[1] Informationen zu Grails sind zu finden unter http://grails.org/.
[2] Informationen zu Django sind zu finden unter http://www.djangoproject.com/.

von Vorteilen gegenüber der klassischen Entwicklung genutzt werden können (siehe hierzu ([Sta07], Kap. 2.2)):

- Die Entwickler benötigen keine speziellen technischen Vorkenntnisse, da sie lediglich das Fachgebiet, das modelliert werden muss, kennen müssen. In diesem Fall besteht das Fachgebiet in der Gestaltung von Räumen sowie der Erfassung und Beschreibung der Exponate.
- Entwicklungszeiten werden deutlich reduziert, da der Quelltext automatisch generiert wird.
- Es wird nur getesteter und dokumentierter Quelltext generiert. Dadurch kann die Optimierung der Software-Qualität erreicht werden.

3.5. Fazit

Die Entwickler einer virtuellen Ausstellung sind in der Regel Geschichtswissenschaftler. Sie haben Kenntnisse über das durch die Ausstellung repräsentierte Fachgebiet. Sie haben jedoch wenig bis gar kein softwaretechnisches Hintergrundwissen. Dementsprechend muss die Software zur Erstellung von virtuellen Ausstellungen für Benutzer konzipiert sein, die zwar Erfahrung mit der Arbeit am Computer haben, aber über kein softwaretechnisches Vorwissen verfügen.

Die Entwicklung einer virtuellen Ausstellung mittels HTML erscheint daher nicht sinnvoll, da die Entwickler über technisches Wissen zur Erstellung von HTML-Dokumenten verfügen müssten. Zudem wäre die Verwaltung der HTML-Dokumente besonders bei größeren Ausstellungen mit einem sehr hohen Administrationsaufwand verbunden. Bei der Verwendung eines Web-Frameworks hingegen müssten die Entwickler einer virtuellen Ausstellung zumindest teilweise über technisches Wissen verfügen, um die Web-Anwendung zu entwickeln. Für die Erfassung der Inhalte in einer Datenbank wären dabei nur Kenntnisse im Umgang mit einem entsprechenden Datenbank-Programm erforderlich. Darüber hinaus müssten bei der Erstellung einer Web-Anwendung viele Funktionen entwickelt werden, die in einem CMS bereits umgesetzt sind. Die Verwendung eines CMS schließlich würde zumindest einen Entwickler mit den notwendigen softwaretechnischen Kenntnissen über das CMS voraussetzen.

Die modellgetriebene Softwareentwicklung vermeidet die zuvor beschriebenen Probleme. Stattdessen können bei ihrer Anwendung die folgenden Vorteile genutzt werden:

- Modellgetriebene Softwareentwicklung ermöglicht eine einfache Umsetzung von neuen virtuellen Ausstellungen ohne Kenntnisse von Web-Technologien.
- Modellgetriebene Softwareentwicklung bietet eine hohe Flexibilität bei der nachträglichen Erweiterung des funktionalen Umfangs der Software zur Erstellung von virtuellen

Ausstellungen. Die Funktionalität kann durch Hinzufügen von weiteren Komponenten erweitert werden.

- Die Optimierung der Software-Qualität wird durch die automatische Generierung von Quelltext ermöglicht.
- Modellgetriebene Softwareentwicklung ermöglicht die Transformation in unterschiedliche Medien, wie beispielsweise statisches HTML, dynamisches HTML oder die Erstellung einer PDF-Datei, durch Hinzufügen entsprechender Komponenten.

Im Rahmen dieser Diplomarbeit wird daher ein Werkzeug entwickelt, das dem Ansatz der modellgetriebenen Softwareentwicklung folgt.

Kapitel 4.

Modellgetriebene Softwareentwicklung

Thema dieser Diplomarbeit ist die modellgetriebene Softwareentwicklung. Hierbei wird aus Modellen automatisch Software erzeugt. Laut Brockhaus ist ein Modell *«ein vereinfachtes Abbild der Wirklichkeit»* ([Bar03], S. 588). Unter der "Wirklichkeit" wird im Rahmen der modellgetriebenen Softwareentwicklung dabei ein System verstanden. Dieses System besteht aus Hardware- und Software-Komponenten. Abbildung 4-1 zeigt das Prinzip der modellgetriebenen Softwareentwicklung.

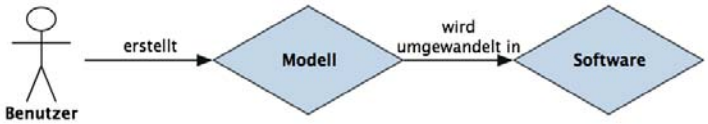


Abb. 4-1. Modellgetriebene Softwareentwicklung

Ziele der modellgetriebenen Softwareentwicklung sind die Steigerung der Softwarequalität und die Steigerung der Effizienz bei der Entwicklung von Software. Dies soll mittels des Prinzips der *Abstraktion* erreicht werden. Abstraktion ist *«das Weglassen von irrelevanten Details»* ([Pet06], S. 45). Für die modellgetriebene Softwareentwicklung bedeutet dies, dass auf einer höheren Abstraktionsebene programmiert wird als es bei der Programmierung mit einer *höheren Programmiersprache* möglich ist. ([Sta07], Kap. 2.2)

Unter einer höheren Programmiersprache (*Hochsprache*) wird hier eine Programmiersprache verstanden, *«deren Programme unabhängig von Prozessoreigenschaften formuliert* [...] *werden können. Programme in höheren* [Programmier-] *Sprachen werden vor der Ausführung in einen Maschinencode gleicher Bedeutung übersetzt.»* ([Bar03], S. 731) Der Maschinencode, auch als *Maschinensprache* bezeichnet, ist dabei prozessorabhängig. Hochsprachen sind dagegen, da sie erst nach der Übersetzung in Maschinencode prozessorabhängige Eigenschaften beinhalten, prozessorunabhängig. Ein Programm, das in einer Hochsprache geschrieben ist, wird somit nur einmal erstellt. Mittels geeigneter Übersetzer wird es dann für die unterschiedlichen Prozessoren in Maschinencode überführt. Ein Programm, das in Maschinensprache

geschrieben ist, muss dagegen für jeden Prozessor neu entwickelt werden. Hochsprachen stehen daher, da bei ihrer Verwendung die prozessorabhängigen Details weggelassen werden, auf einer höheren Abstraktionsebene als Maschinencode. Des Weiteren ist der Quelltext eines in einer Hochsprache geschriebenen Computerprogramms leichter lesbar als ein Quelltext, der in Maschinencode geschrieben ist. Dies ist darin begründet, dass Hochsprachen der natürlichen Sprache ähnlicher sind als Maschinencode. Die Entwicklung eines Computerprogramms mittels einer Hochsprache ist daher einfacher als die Entwicklung eines Computerprogramms in Maschinensprache. ([Bar03], S. 730f.)

Hochsprachen wie Java oder C# abstrahieren darüber hinaus von einem konkreten Betriebssystem. Bei der Programmierung mit Java beispielsweise wird durch den *Java Compiler* aus dem Quelltext *Java Bytecode* erzeugt. Unter Bytecode wird hier *«die Kodierung eines Computerprogramms* [...] *in einer abstrakten, prozessorunabhängigen Form* [...]*»* ([Bar03], S. 143) verstanden. Der Java Bytecode wird bei der Ausführung des Programms durch die *Java Virtual Machine* interpretiert. Dasselbe Java-Programm kann somit, wenn die Java Virtual Machine installiert ist, auf verschiedenen Betriebssystemen ausgeführt werden. ([Bar03], S. 486f.)

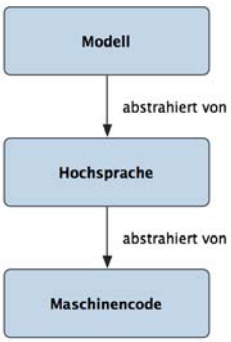


Abb. 4-2. Abstraktionsstufen

Bei der modellgetriebenen Softwareentwicklung wird, wie in Abbildung 4-2 dargestellt, durch das Modell von der konkreten (höheren) Programmiersprache abstrahiert. Im Modell werden dazu implementierungsspezifische Details weggelassen. Der Entwickler konzentriert sich auf die Funktionalität der zu entwickelnden Anwendung ([Atk03]). Die Software kann somit *plattformunabhängig* entwickelt werden. Plattformunabhängig bedeutet in diesem Zusammenhang, dass die Anwendungslogik weder auf eine Programmiersprache oder ein Framework, noch auf ein bestimmtes Betriebssystem festgelegt ist. Des Weiteren werden Routinearbeiten bei der Implementierung dem Entwickler abgenommen und automatisch ausgeführt. Es können beispielsweise automatisch Tests für die Software generiert werden. Dies kann zu einer Zeitersparnis gegenüber der Softwareentwicklung ohne modellgetriebenen Ansatz führen. Darüber hinaus werden Fehler minimiert, da nur getesteter und dokumentierter Quellcode automatisch erzeugt wird ([Sta07], Kap. 2.2).

In diesem Kapitel werden zunächst die relevanten Begriffe und Konzepte der modellgetriebenen Softwareentwicklung erläutert. Im Anschluss wird auf *Model Driven Architecture* (MDA), eine spezielle Ausprägung von modellgetriebener Softwareentwicklung eingegangen.

4.1. Model Driven Development

Modellgetriebene Softwareentwicklung wird als *Model Driven Development* (MDD) oder *Model Driven Software Development* (MDSD) bezeichnet. ([Tro07], S. 12)

> *«Modellgetriebene Softwareentwicklung (Model Driven Software Development, MDSD) ist ein Oberbegriff für Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugen.»* ([Sta07], S. 11)

Zunächst muss für das Verständnis dieser Definition geklärt werden, was ein *Modell* ist. Im Anschluss daran werden die Konzepte der Erstellung von Modellen, der sogenannten *Modellierung*, erläutert. Schließlich wird die Erzeugung von lauffähiger Software aus einem Modell dargestellt. Dieser Prozess wird als *Generierung* bezeichnet.

4.2. Modell

«Ein Modell stellt ein Abbild eines realen oder abstrakten Systems dar [...].*»* ([Pet06], S. 42) Im Rahmen dieser Diplomarbeit wird die Software-Komponente dieses Systems betrachtet. Im Folgenden werden dabei die Begriffe "System" oder "Software" für die Software-Komponente synonym verwendet. Ein Modell wird hier als Vorbild aufgefasst, das der Beschreibung des Systems dient ([Sch99]). Die Problematik der unterschiedlichen Auffassungen des Modellbegriffs bezüglich der Rolle des Modells als Vor- oder Nachbild eines Systems wird von Schefe ([Sch99]) ausführlich diskutiert.

Modelle gibt es in unterschiedlichen Ausprägungen. Ein Modell kann beispielsweise eine Skizze sein, ein UML-Diagramm oder es kann in textueller Form vorliegen. Es kann menschen- und/oder maschinenlesbar sein. ([Pet06], Kap. 2.2)

Für die modellgetriebene Softwareentwicklung muss ein Modell *formal* sein. "Formal" bedeutet, dass sich das Modell an ein Regelwerk hält, *«das mathematisch exakt, widerspruchsfrei und vollständig definiert»* ([Pet06], S. 42) ist. Ein formales Modell beschreibt einen Teil einer Software vollständig. Dieser Teil kann technisch oder fachlich sein. Ein technischer Teil wäre beispielsweise die Architektur der Software. Ein fachlicher Teil bezieht sich hingegen auf die Fachlogik der Anwendung. ([Sta07], Kap. 2.1)

Zweck eines Modells ist bei der modellgetriebenen Softwareentwicklung nicht nur die Dokumentation und Beschreibung der Software. Aus einem Modell wird hierbei auch automatisch die ausführbare Anwendung erzeugt ([Tro07], Kap. 1.3). Darüber hinaus ist das Ziel bei der Erstellung eines Modells, die Komplexität des Modells gegenüber der Komplexität der Software zu reduzieren, um deren Entwicklung zu vereinfachen. Eigenschaften, die nicht zu dem durch das Modell abgebildeten Teil der Software gehören, werden dafür weggelassen ([Pet06], Kap. 2.2).

Im Zusammenhang mit dem Modellbegriff stehen weitere Begriffe, die in Abbildung 4-3 dargestellt sind. Im Folgenden werden diese Begriffe erläutert.

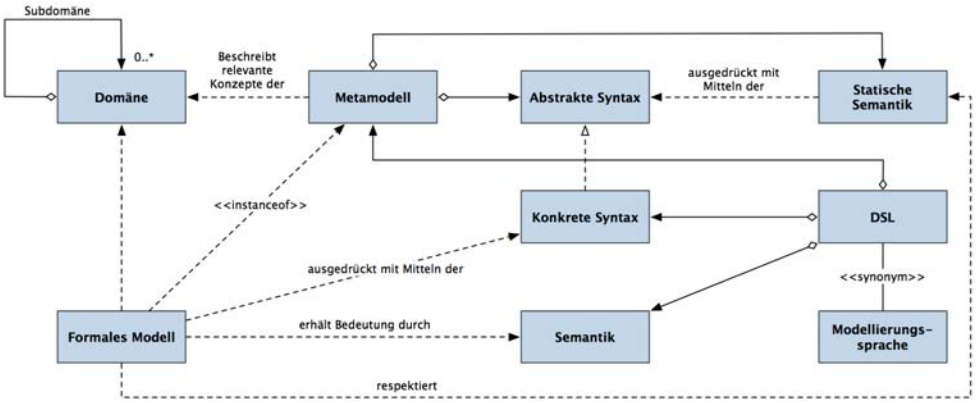


Abb. 4-3. Begriffe der MDD, Quelle: ([Sta07], S. 28)

4.2.1. Domäne

Eine *Domäne* ist ein begrenztes Wissens- oder Interessengebiet. Sie kann technisch oder fachlich sein. Eine technische Domäne bezieht sich auf den technischen Teil einer Software. Eine fachliche Domäne bezieht sich auf den fachlichen Teil einer Software. Eine Domäne kann aus Subdomänen bestehen. Eine Subdomäne ist eine Teilmenge der Domäne. Subdomänen sind beispielsweise sinnvoll für den Umgang mit sehr komplexen Domänen, um die Überschaubarkeit der Domäne zu gewährleisten. ([Sta07], Kap. 3.1)

4.2.2. Domänenspezifische Sprachen

Eine *domänenspezifische Sprache* wird als *Domain Specific Language* (DSL) bezeichnet. Sie ist eine speziell auf eine Domäne abgestimmte Sprache ([Sta07], Kap. 3.1). Die Sprachelemente der DSL werden von Mensch und Maschine verstanden. Des Weiteren wird mittels der DSL ein formales Modell erstellt. Der Mensch kann somit den Sachverhalt einer Domäne maschinenverständlich abbilden ([Tro07], Kap. 3.6).

Im Wesentlichen umfasst eine DSL ein *Metamodell* sowie eine *abstrakte* und eine *konkrete Syntax*. Das Metamodell und die abstrakte Syntax dienen hierbei der Abbildung der Struktur der durch die DSL beschriebenen Domäne. Die konkrete Syntax hingegen legt die Art der Darstellung der DSL fest. ([Sta07], Kap. 3.1).

4.2.3. Metamodell

Ein Metamodell macht Aussagen über die mögliche Struktur eines Modells. Es ist die formale Beschreibung einer Domäne ([Sta07], Kap. 3.1). Die Vorsilbe "meta" bedeutet hier soviel wie "über" oder "übergeordnet". Die einzelnen Elemente eines Modells sind dabei Instanzen der Modellelemente des Metamodells. Das heißt, das Metamodell steht auf einer höheren Ebene als das Modell ([Pet06], Kap. 2.2). Zur Veranschaulichung der Beziehung zwischen Domäne, Modell und Metamodell siehe Abbildung 4-4.

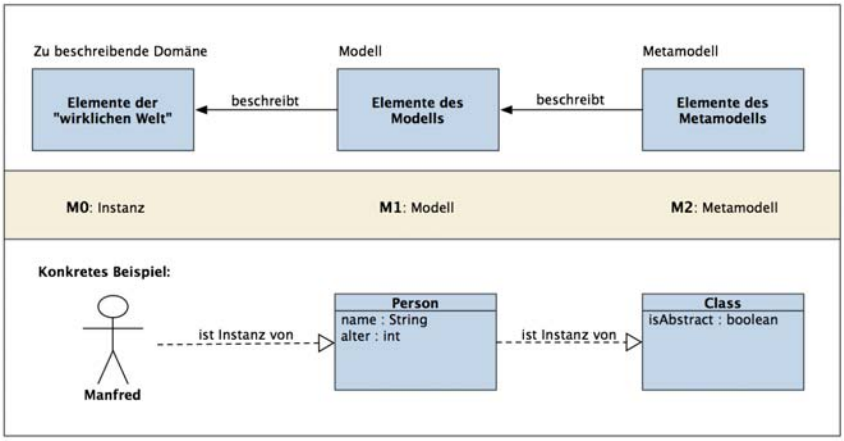


Abb. 4-4. (Meta-)Modellebenen, Quelle: angelehnt an ([Sta07], S. 60) und ([Pet06], S. 49)

Neben dem Metamodell gibt es das *Domänenmodell*. Im Rahmen der UML definiert Larman den Begriff "Domänenmodell" wie folgt:

> *«A domain model is a visual representation of conceptual classes or real-world objects in a domain of interest.»* ([Lar01], Kap. 10.1)

Darüber hinaus bildet ein Domänenmodell die Beziehungen der Elemente der Domäne untereinander ab. Das Domänenmodell beschreibt also ähnlich wie das Metamodell die Domäne einer Anwendung.

In der Literatur zur modellgetriebenen Softwareentwicklung wird das Verhältnis beziehungsweise der Unterschied zwischen Metamodell und Domänenmodell nicht definiert. Im Rahmen dieser Diplomarbeit wird das Domänenmodell als die Beschreibung der Domäne entsprechend der Definition von Larman aufgefasst. Des Weiteren wird davon ausgegangen, dass das Metamodell auf Basis des Domänenmodells durch ein geeignetes Werkzeug im Kontext der modellgetriebenen Softwareentwicklung erstellt wird. Dabei weist es die oben beschriebenen Eigenschaften auf.

4.2.4. Metametamodell

Ein *Metametamodell* ist das Metamodell eines Metamodells. Die Modellelemente des Metamodells sind also Instanzen der Modellelemente des Metametamodells. In der Theorie kann es beliebig viele Meta-Ebenen für ein Modell geben. In der Praxis sind drei Ebenen in den meisten Fällen ausreichend: Modell, Metamodell und Metametamodell. Das Metametamodell beschreibt sich dabei meist selbst. Ein weit verbreitetes Metametamodell ist die *Meta Object Facility* (MOF), ein Standard der *Object Management Group* (OMG). ([Sta07], Kap. 3.1)

Die Bezeichnungen "Metamodell" und "Metametamodell" sind nicht absolut. Ein Metametamodell bezogen auf ein Modell ist gleichzeitig das Metamodell des Metamodells des Modells. Das Metamodell des Modells wiederum ist selbst ein Modell bezogen auf das Metametamodell. Die MOF beispielsweise ist unter anderem das Metamodell der *Unified Modeling Language* (UML) und das Metametamodell der Modelle, die mittels der UML erstellt werden. ([Sta07], Kap. 5.1)

4.2.5. Abstrakte und konkrete Syntax

Eine DSL hat zwei Syntaxen. Die abstrakte Syntax und die konkrete Syntax. Die abstrakte Syntax beschreibt die Metamodellelemente und ihre Beziehungen untereinander ([Sta07], Kap. 3.1). Allgemeiner ausgedrückt definiert die abstrakte Syntax die syntaktischen Elemente, zum Beispiel Buchstaben, und legt fest, wie daraus Konstrukte, zum Beispiel Wörter, gebildet werden. Ein Konstrukt ist dabei eine Kombination von syntaktischen Elementen. ([Pet06], Kap. 1.3). Die abstrakte Syntax der Programmiersprache Java beispielsweise legt fest, dass es Klassen gibt, die einen Namen haben und die Methoden haben können. ([Sta07], Kap. 3.1).

Die konkrete Syntax hingegen legt fest, wie ein Modell oder allgemein gesagt ein Konstrukt konkret beschrieben wird. Sie legt die *«Ausdrucksmittel (Darstellungsform, Notation)»* ([Pet06], S. 19) der DSL fest. Für die Programmiersprache Java legt die konkrete Syntax beispielsweise fest, dass eine Klasse mit dem Schlüsselwort "class" beginnt. Des Weiteren definiert sie, dass die Implementierung eines Interface durch eine Klasse mittels des Schlüsselworts "implements" gekennzeichnet ist. ([Sta07], Kap. 3.1)

Eine konkrete Syntax kann graphisch oder textuell sein. Ein Beispiel für eine graphische konkrete Syntax ist die UML-Notation. Eine textuelle konkrete Syntax ist die Notation in XML. Entsprechend der Art der konkreten Syntax wird ein Parser benötigt. Dieser liest ein Modell ein und wandelt es hinsichtlich der abstrakten Syntax in ein anderes Format um. Das andere Format dient dabei der Verarbeitung des Modells. Der Parser kann beispielsweise ein Modell, das mit XML als konkreter Syntax erstellt wurde, auf miteinander verbundene Java-Objekte abbilden. Die abstrakte Syntax würde hierbei unter anderem durch die entsprechenden Java-Klassen beschrieben werden. ([Sta07], Kap. 3.1)

Petrasch et al. machen darüber hinaus die folgende Aussage zur konkreten und abstrakten Syntax.

> *«Modelle und Metamodelle können dieselbe konkrete Syntax verwenden, besitzen jedoch eine unterschiedliche abstrakte Syntax bzw. Semantik.»* ([Pet06], S. 50)

Diese Aussage berücksichtigt jedoch nicht den Fall, dass ein Modell sich selbst beschreibt. Die MOF beispielsweise ist, da sie sich selbst beschreibt, ihr eigenes Metamodell. Modell und Metamodell haben in diesem Fall daher dieselbe konkrete Syntax und dieselbe abstrakte Syntax.

4.2.6. Statische und dynamische Semantik

Die *statische Semantik* legt die Bedingungen fest, die ein Modell erfüllen muss, um wohlgeformt zu sein. Ein Modell, das wohlgeformt ist, wird auch als valide bezeichnet. Die Bedingungen der statischen Semantik werden *Constraints* genannt ([Sta07], Kap. 3.1). Allgemein gesagt, legt die statische Semantik fest, wie ein Konstrukt mit einem anderen verbunden ist, um eine Bedeutung zu erhalten. Die statische Semantik der booleschen Algebra beispielsweise legt fest, dass ein boolescher Ausdruck der Form "boolescher Wert, boolescher Operator, boolescher Wert" einen booleschen Wert zurückliefert ([Pet06], Kap. 1.3). Bei statisch getypten Programmiersprachen sind die Bedingungen, die vom Compiler geprüft werden, die statische Semantik. Diese legen beispielsweise fest, dass Variablen deklariert sein müssen ([Sta07], Kap. 3.1).

Die *dynamische Semantik* hingegen legt die erlaubten Ausführungen des durch die statische Semantik definierten Konstrukts fest. Nur wenn ein Konstrukt wohlgeformt ist, definiert die dynamische Semantik dabei eine Bedeutung. Die dynamische Semantik legt beispielsweise fest, dass ein boolescher Ausdruck der Form "boolescher Wert, boolescher Operator, boolescher Wert" bezogen auf den UND-Operator nur dann zu "true" evaluiert, wenn beide booleschen Werte "true" sind, sonst evaluiert er zu "false" ([Pet06], Kap. 1.3). Wird bei der Programmierung gegen die dynamische Semantik verstoßen, wird dies erst zur Laufzeit erkannt. Das folgende Beispiel verdeutlicht dies.

```
1 float i = x;
2 i = i - 1;
3 i = 1/i;
```

Listing 4-1 Dynamische Semantik, Quelle: angelehnt an ([Sch96], Kap. 10.1.3)

Die potentielle Division durch Null wird erst zur Laufzeit erkannt. Da eine Division durch Null gegen die dynamische Semantik verstößt, kommt es für den Fall, dass x den Wert 1 hat, zu einem Fehler. ([Sch96])

4.3. Modellierung

Bei der Anwendung des Ansatzes der modellgetriebenen Softwareentwicklung muss, da die Elemente eines Modells Instanzen der Elemente des Metamodells sind, mit der Definition des Metamodells begonnen werden. Das Metamodell bildet die Grundlage für die Modellierung. Die Schwierigkeit bei der Definition des Metamodells besteht darin, es vollständig zu definieren. Die Vollständigkeit des Metamodells ist für die Modellierung von Bedeutung, da ein Modell die gesamten relevanten Eigenschaften des abzubildenden Teils eines Systems umfassen können muss. Zu diesem Zweck sollten mehrere konkrete Szenarien bezüglich der Eigenschaften des Systems vor und während der Erstellung des Metamodells durchdacht werden. ([Sta07], Kap. 5.5)

Ein Metamodell, das alle relevanten Eigenschaften eines Systems beschreibt, kann sehr umfangreich und dadurch sehr unübersichtlich werden. Es bietet sich daher an, ein solches Metamodell auf mehrere Metamodelle aufzuteilen. Der Bereich, den jedes Metamodell dabei beschreibt, ist eindeutig abgegrenzt. Es kann beispielsweise ein Metamodell geben, das die Geschäftslogik der einen Komponente des Systems abbildet und ein zweites, das die Geschäftslogik einer anderen Komponente beschreibt. Das Metamodell des gesamten Systems besteht dann aus einer Reihe von einzelnen Metamodellen. Ein solches Metamodell wird als *modulares Metamodell* bezeichnet. ([Sta07], Kap. 5.3)

Für die Erstellung von Modellen auf Basis des Metamodells wird eine konkrete Syntax benötigt. Die konkrete Syntax ist für die Verarbeitung eines Modells, zum Beispiel bei der Generierung, weitgehend irrelevant. Sie ist jedoch die Schnittstelle zum Modellierer. Ihre Qualität ist damit entscheidend für die Lesbarkeit eines Modells. Es ist dabei durchaus möglich, dass es mehrere konkrete Syntaxen, zum Beispiel eine graphische und eine textuelle, für ein Modell gibt. ([Sta07], Kap. 5.1)

4.3.1. Modellvalidierung

Bei der Modellierung besonders von komplexen Modellen kommt es häufig zu *Modellierungsfehlern*. Modellierungsfehler sind Verstöße gegen die abstrakte Syntax oder gegen die durch die statische Semantik festgelegten Constraints. Je später diese Verstöße erkannt werden, desto höher ist in der Regel der Aufwand sie zu beheben ([Tro07], Kap. 3.7). Teil der modellgetriebenen Softwareentwicklung ist daher die *Modellvalidierung* (*Validierung*). Bei der Modellvalidierung wird geprüft, ob die abstrakte Syntax und die Constraints vom Modell eingehalten werden. Die Software oder der Teil der Software, der die Validierung durchführt, wird dabei als *Validator* bezeichnet. ([Sta07], Kap. 5.1).

Das Zusammenspiel von Modellierung und Validierung ist ein iterativer Prozess. Durch die Validierung entdeckte Fehler werden durch erneute Modellierung behoben ([Tro07], Kap. 3.7). Abbildung 4-5 verdeutlicht den Modellierungs- und Validierungsprozess. Bei der Generierung

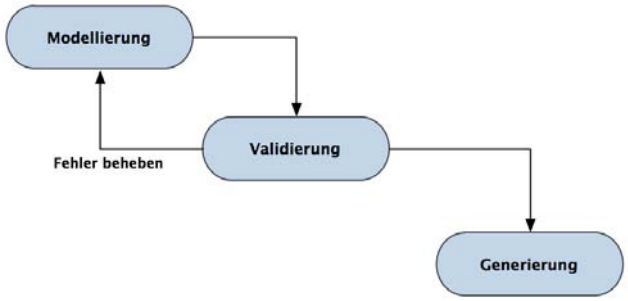


Abb. 4-5. Modellierung und Validierung, Quelle: ([Tro07], S. 67)

wird meistens vorausgesetzt, dass das Modell valide ist. Ist das Modell nicht valide, kann es zur Generierung von fehlerhaftem Quelltext kommen. Die Modellvalidierung sollte daher mindestens direkt vor der Generierung ausgeführt werden. Darüber hinaus kann die Validierung beispielsweise während der Eingabe des Modells stattfinden. Der Zeitpunkt der Eingabe ist der frühestmögliche Zeitpunkt für die Validierung. Sie wird dabei vom Editor durchgeführt. Diese Technik der "Validierung während der Eingabe" ist weit verbreitet und in den meisten modernen integrierten Entwicklungsumgebungen, den sogenannten *Integrated Development Environments* (IDE), umgesetzt. Bei der Validierung im Editor und bei der Validierung direkt vor der Codegenerierung sollte auf die gleiche Implementierung der Constraints zurückgegriffen werden, um die Konsistenz in der Validierung zu gewährleisten. ([Sta07], Kap. 5.1)

4.3.2. Konzepte von domänenspezifischen Sprachen

Domänenspezifische Sprachen können nach unterschiedlichen Kriterien kategorisiert werden. Sie können in interne und externe DSLs oder in graphische und textuelle DSLs eingeteilt werden. Die Wahl der DSL und damit der konkreten Syntax ist von Bedeutung, da von ihr unter anderem die Wahl des Modellierungswerkzeugs und die Lesbarkeit des Modells abhängen. ([Sta07], Kap. 6)

Martin Fowler hat die Begriffe der *internen DSL*, auch *embedded DSL*, und *externen DSL* geprägt ([Fow05]). Eine interne DSL ist eine DSL, die in eine andere Programmiersprache, die sogenannte *Hostsprache*, eingebettet ist, daher der Begriff der embedded DSL. Die Hostsprache ist meist eine dynamisch getypte Sprache wie zum Beispiel *Ruby*. Hierbei werden beispielsweise neue Sprachkonstrukte mit den Mitteln der Hostsprache definiert. Mit diesen kann dann domänenspezifisch programmiert werden. Eine externe DSL wird dagegen von Grund auf neu konzipiert. Für sie wird ein spezieller Parser benötigt, der die Information der DSL einliest und zur Generierung bereitstellt. ([Sta07], Kap. 6.1)

Eine interne DSL hat verschiedene Nachteile. Zum einen kann eine interne DSL immer nur eine textuelle konkrete Syntax haben. Des Weiteren ist die Entwicklung spezieller Modellierungs-

werkzeuge für die DSL mit hohem Aufwand verbunden, da die DSL nur durch die Hostsprache begrenzt wird. Insbesondere bei der Verwendung einer dynamisch getypten Programmiersprache ist darüber hinaus die Entwicklung eines statischen Validators für die DSL schwer möglich, da dabei die Typzuweisung einer Variablen erst zur Laufzeit geschieht. Modellierungsfehler werden somit erst spät oder gar nicht erkannt ([Sta07], Kap. 6.1). Ein weiterer Nachteil ist, dass dem Entwickler, der mit der DSL ein Modell erstellen möchte, mehr Möglichkeiten zur Verfügung stehen als für die Modellierung gebraucht werden. Dies erschwert die Modellierung besonders dann, wenn der Entwickler die Hostsprache nicht beherrscht. Vorteil einer internen DSL ist, dass ein Entwickler, der die Hostsprache dagegen gut kennt, die volle Mächtigkeit der Hostsprache ausnutzen kann ([Fow05]).

Nachteil externer DSLs ist, dass der Editor und Parser für die DSL zusätzlich zur DSL entwickelt werden muss. Darin liegt jedoch auch gleichzeitig der Vorteil. Es kann beispielsweise ein graphischer Editor für die DSL entwickelt werden. Des Weiteren kann ein solcher Editor so implementiert werden, dass er im Gegensatz zu einem Editor einer internen DSL nur die Funktionalität bereitstellt, die zur Erstellung des Modells benötigt wird. ([Fow05])

Ein ausführlicher Vergleich von internen und externen DSLs findet sich bei Fowler in dem Abschnitt “Pros and Cons of language oriented programming” ([Fow05]). Ein Beispiel für die Entwicklung einer internen DSL unter Verwendung von Ruby findet sich bei Cuadrado et al. ([Cua07]).

Eine weitere Kategorisierung von DSLs ist deren Unterteilung in graphische und textuelle DSLs. Eine graphische DSL besitzt eine graphische konkrete Syntax. Bei Modellen mit einer graphischen DSL werden die Modellelemente graphisch dargestellt. Solche Modelle geben meist einen guten Überblick über ihre Struktur. Sie sind damit in der Regel als Diskussionsgrundlage gut geeignet. Dabei muss jedoch beachtet werden, nicht zu viele Elemente oder Konzepte im Modell abzubilden, um die Übersichtlichkeit beizubehalten. Textuelle DSLs besitzen eine textuelle konkrete Syntax. Modelle mit einer textuellen DSL sind in der Regel weniger übersichtlich als Modelle mit einer graphischen DSL. Dies kann durch entsprechende Hilfsmittel des benutzten Editors, wie zum Beispiel Syntax-Highlighting, verbessert werden. Als Diskussionsgrundlage sind sie jedoch in der Regel ungeeignet. ([Sta07], Kap. 6.1)

4.4. Transformation

Bei einer Transformation wird *«aus einem formalen Modell etwas anderes erzeugt»* ([Sta07], S. 33). Durch die Transformation kann Text erzeugt werden wie zum Beispiel Quelltext oder Dokumentationsdateien, oder es kann ein weiteres Modell erzeugt werden. Wird aus einem Modell Text erzeugt, spricht man von einer *Modell-zu-Code-Transformation* (M2C-Transformation). Wird aus einem Modell ein Modell erzeugt, wird dies *Modell-zu-Modell-Transformation* (M2M-Transformation) genannt. ([Sta07], Kap. 3.1)

Für die Durchführung einer Transformation wird eine spezielle Software benötigt. Diese wird als *Codegenerator* oder *Generator* bezeichnet. Ein Generator liest ein oder mehrere Modelle ein, validiert sie gegebenenfalls und führt dann eine oder mehrere Transformationen durch. Dieser Vorgang wird *Generierungsprozess* oder *Transformationsprozess* genannt. Hierfür muss ein Generator die folgenden drei grundlegenden Funktionen unterstützen.

- Er ermöglichst das Schreiben von Text in Dateien.
- Er ermöglichst die Abfrage von Werten aus einem eingegebenen Modell.
- Die Konkatenation von statischem Text, das heißt statischen Informationen, und Informationen aus dem Modell wird unterstützt.

Ein Generator, der im Rahmen der modellgetriebenen Softwareentwicklung angewandt wird, unterscheidet darüber hinaus zwischen zwei Arten von Informationen: variable Informationen und statische Informationen. Variable Informationen werden durch das Modell festgelegt und sind je nach Modell unterschiedlich. Statische Informationen werden durch das oder die zu verarbeitenden Modelle nicht beeinflusst. Sie sind die sogenannten *Generatorvorschriften*. Die Generatorvorschriften sind für die Modellierung irrelevant und werden erst bei der oder den Transformationen hinzugezogen ([Sta07], Kap. 8.3). Des Weiteren definieren Stahl et al. für einen Generator innerhalb der modellgetriebenen Softwareentwicklung die folgenden Merkmale (siehe hierzu ([Sta07], S. 145)):

- Die variablen Informationen (das Modell) sind von den statischen Informationen (die Generatorvorschriften) getrennt.
- Die Generatorvorschriften basieren auf dem Metamodell.
- Der Generierungsprozess kann beliebig oft für ein Modell durchgeführt werden.
- Ein Generator kann für beliebig viele Modelle verwendet werden. Die Modelle müssen jedoch alle dem Metamodell entsprechen, auf dem die Generatorvorschriften basieren.

4.4.1. Modell-zu-Code-Transformation

Bei der M2C-Transformation werden aus Modellen Quelltexte und weitere Dateien erzeugt. Diese weiteren Dateien sind in der Regel Dateien, die zur Ausführung der zu generierenden Software erforderlich sind. Sie können beispielsweise Konfigurationsdateien oder Datenbankskripte sein. Bei der M2C-Transformation können jedoch auch nicht zur Ausführung der Software notwendige Dateien, zum Beispiel Dokumentationsdateien, generiert werden. Die M2C-Transformation wird daher auch als *Modell-zu-Text-Transformation* bezeichnet. Eine weitere Bezeichnung ist *Codegenerierung*, kurz *Generierung*. Diese setzt ebenso wie der Begriff "M2C-Transformation" den Schwerpunkt auf die Erzeugung von Quelltext. Die unterschiedlichen Bezeichnungen für die M2C-Transformation werden in der Literatur weitgehend

synonym verwendet. Im Folgenden wird hauptsächlich der Begriff "Codegenerierung" beziehungsweise "Generierung" verwendet. Abbildung 4-6 zeigt eine schematische Darstellung der M2C-Transformation.

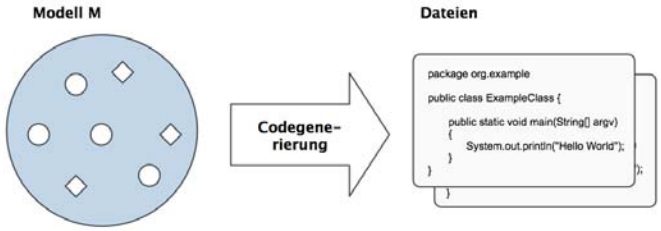


Abb. 4-6. Modell-zu-Code-Transformation

Der durch eine M2C-Transformation erzeugte Quelltext ist spezifisch für eine *Plattform*. Unter einer Plattform werden hier zum einen die Hardware- und Software-Komponenten eines Rechners verstanden. Zum anderen umfasst der Begriff "Plattform" Programmbibliotheken und Middleware wie beispielsweise J2EE oder CORBA (Common Object Request Broker Architecture). Bei einer M2C-Transformation können beispielsweise Java-Klassen und spezielle Konfigurationsdateien erzeugt werden, die spezifisch für eine J2EE-Anwendung sind. Die Plattform, für die die Dateien erzeugt werden, wird dabei als *Zielplattform* oder *Zielsystem* bezeichnet.

Für die Codegenerierung gibt es unterschiedliche Techniken. Diese setzen unterschiedliche Schwerpunkte bezüglich beispielsweise der Konkatenation von statischem Text und Informationen aus dem Modell. Im Folgenden werden zwei mögliche Techniken der Codegenerierung vorgestellt. Zunächst wird die Codegenerierung mit Hilfe von Templates erläutert. Im Anschluss wird auf die Codegenerierung mittels einer Programmiersprache eingegangen.

Die Codegenerierung mittels Templates ist sinnvoll, wenn viel statischer Text generiert werden soll. Der Generator verwendet hierbei eine *Template-Engine*. Eine Template-Engine ist eine Software, die Dateien anhand von Vorlagen, sogenannten *Templates*, erstellt. Ein Template besteht aus beliebigem Text, der *Tags* beinhaltet. Ein Tag ist dabei ein Platzhalter in einer speziellen Notation, der während des Generierungsprozesses von der Template-Engine durch einen entsprechenden Wert ersetzt wird. Die Notation der Tags wird durch die *Template-Sprache* spezifiziert. Die Template-Sprache legt beispielsweise fest, wie ein Template aufgebaut ist oder wie ein Tag notiert wird.

Ein Tag enthält einen Ausdruck, der durch die Template-Engine ausgewertet wird. Das Ergebnis der Auswertung wird an die Stelle des Tags gesetzt. Ein Ausdruck kann zum einen dazu dienen, Informationen aus dem Modell abzufragen. Zum anderen kann er den Ablauf der Verarbeitung des Templates durch die Template-Engine mittels Bedingungen oder Schleifen steuern. Er dient in diesem Fall als Kontrollstruktur. Mit Hilfe eines Tags, das eine solche Kontrollstruktur enthält, kann zum Beispiel eine Datei generiert werden, die eine Liste enthält,

deren Länge durch das Modell festgelegt wird. Die Form eines Ausdrucks ist ebenso wie die Notation der Tags von der Template-Sprache abhängig. Template-Sprachen sind beispielsweise *JavaServer Pages* (JSP) oder aus dem XML-Umfeld XSLT. Die Technik der Codegenerierung mittels Templates ist spezialisiert auf die Konkatenation von statischem Text und Modellinformationen. ([Sta07], Kap. 8.4.1)

Bei der Codegenerierung mittels einer Programmiersprache wird der Generator mit Hilfe einer Programmiersprache wie beispielsweise Java oder C# entwickelt. Meistens werden dabei Klassen erstellt, die bestimmte Konzepte der *Zielsprache* repräsentieren. Unter Zielsprache wird hier die Programmiersprache, beispielsweise Java, oder Auszeichnungssprache, beispielsweise XML, verstanden, in der die zu generierenden Dateien verfasst sind. Bei dieser Art der Codegenerierung würde beispielsweise eine Klasse das Klassen-Konzept in Java repräsentieren. Eine solche Klasse befindet sich in Anhang A.1. Sie wäre für die Generierung von Java-Klassen verantwortlich. Diese Klasse enthielte zum Beispiel eine Methode, wie sie in Listing 4-2 gezeigt wird. Diese Methode erzeugt den Quelltext für eine Java-Klasse, der bei der Codegenerierung in eine Datei geschrieben wird.

```
1 public String toString()
2 {
3     String buff = "package " + packageName + ";\n" +
4         "\n" +
5         "public class " + name + " {\n";
6     /* ... */
7     return buff;
8 }
```

Listing 4-2 Methode zur Erstellung von Java-Klassen, Quelle: ([Sta07], S. 150)

Bei dieser Technik der Codegenerierung fließen die Eigenheiten der gewählten Programmiersprache in die Entwicklung des Generators mit ein, was unter Umständen zu Problemen führen kann. In Listing 4-2 wird beispielsweise mit Strings in Java gearbeitet, was Nachteile mit sich bringt. Es gibt zum Beispiel in Java keine mehrzeiligen Strings. Zeilenumbrüche müssen in Java mit Escape-Sequenzen, beispielsweise "\n", beschrieben werden. Des Weiteren können Strings im Quelltext nicht über mehrere Zeilen gehen, sie müssen mit "+" konkateniert werden. Diese Eigenschaften von Java machen den Quelltext, wenn viel statischer Text erzeugt wird, schnell unübersichtlich. Der Vorteil der Codegenerierung mittels einer Programmiersprache ist, dass die volle Unterstützung der meist sehr leistungsstarken IDEs, wie beispielsweise Syntax-Highlighting und Debugging-Funktionen, für die Entwicklung des Generators genutzt werden kann. ([Sta07], Kap. 8.4.2)

4.4.2. Modell-zu-Modell-Transformation

In der *«'klassischen' MDSD»* ([Sta07], S. 196) folgt nach der Modellierung und Validierung des Modells die Codegenerierung. Das bedeutet, dass es zwei Abstraktionsebenen gibt: die Ebene

des Modells und die Ebene des Quelltextes. In den meisten Fällen ist dies ausreichend. Unter Umständen wird der Codegenerator bei diesem Ansatz jedoch sehr komplex, da er bei der Codegenerierung die gesamten Abstraktionen des Modells konkretisieren muss. Insbesondere bei Modellen, die sehr stark vom Code abstrahieren, kann die Erzeugung des Codes dabei sehr umfangreich sein. Dieses Problem soll durch das folgende Beispiel verdeutlicht werden. Für jedes Modellelement A soll eine entsprechende Klasse Ai erzeugt werden, wobei i einen Wert zwischen 1 und 9 einnimmt. Des Weiteren soll eine Klasse B generiert werden, die für jede Klasse Ai eine Methode zur Erzeugung dieser Klasse Ai besitzt. Diese Aufgaben können entweder durch den Modellierer gelöst werden oder durch den Codegenerator. Im ersten Fall würde der Modellierer zum einen in den Elementen des Modells, die vom Typ A sind, die Namen der zu generierenden Klassen angeben. Zum anderen würde er im Modell ein Element B erstellen und die erforderlichen Methoden hinzufügen. Würden die Aufgaben durch den Codegenerator gelöst werden, würde dieser entsprechend komplexer werden, da er um die dafür erforderliche Logik erweitert werden müsste. ([Sta07], Kap. 10.1)

Ein weiteres Problem besteht darin, dass bei der Generierung dasselbe Metamodell wie bei der Modellierung verwendet wird. Die für die Generierung notwendigen Konzepte müssen somit in das Metamodell mit aufgenommen werden, wodurch das Metamodell möglicherweise zu sehr "aufgebläht" wird. Das zuvor angeführte Beispiel verdeutlicht auch dieses Problem. Element B gehört unter Umständen zum technischen Teil der zu generierenden Software. Element A hingegen gehört zum fachlogischen Teil. Das Element B müsste in das Metamodell aufgenommen werden, um modelliert werden zu können. Dies sollte jedoch insbesondere im Hinblick auf die unterschiedlichen Aspekte der Software, die die Elemente beschreiben, vermieden werden. ([Sta07], Kap. 10.1)

Auf Grund dieser Probleme ist daher ein weiterer Schritt erforderlich, der in der Regel zwischen der Modellierung und der Codegenerierung liegt: die Modell-zu-Modell-Transformation. In diesem Schritt wird das Modell für die Codegenerierung "aufbereitet". Diese Aufbereitung ist eine Anreicherung des Modells mit zusätzlichen Informationen (die *Modellmodifikation*) oder die Überführung des Modells in ein anderes Modell (die *Modelltransformation*). ([Sta07], Kap. 10.1)

Bei der Modellmodifikation wird das Ausgangsmodell modifiziert. Das Ausgangsmodell ist das Modell, auf welches die Transformation angewandt wird. Abbildung 4-7 verdeutlicht diese Art der Modell-zu-Modell-Transformation. Das Modell wird dabei mit Informationen angereichert. Hierfür werden basierend auf den Metamodellelementen Regeln definiert, wie die Modellelemente mit Informationen anzureichern sind ([Tro07], Kap. 3.8.1). Das so entstandene Modell hat weiterhin dasselbe Metamodell wie das Ausgangsmodell. In der Abbildung ist dies durch die gleiche geometrische Form der Modellelemente beider Modelle dargestellt. Dem Ausgangsmodell werden neue Instanzen der Metamodellelemente hinzugefügt und bestehende verändert. In der Abbildung ist dies durch eine andere Färbung verdeutlicht ([Sta07], Kap. 10.2).

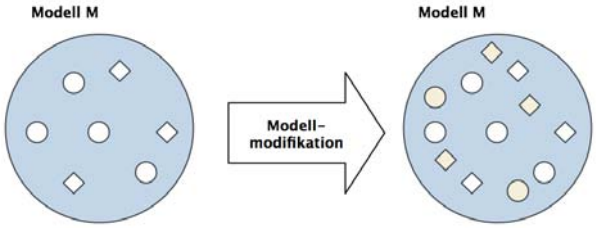


Abb. 4-7. Modellmodifikation, Quelle: ([Sta07], S. 199)

Als einen Spezialfall der Modellmodifikation führen Stahl et al. das sogenannte *Modell-Weaving* (*Linking*) an. In manchen Fällen ist es notwendig, nicht alle Informationen in einem Modell zu speichern. Die Informationen sind dann auf mehrere Modelle verteilt. In diesem Fall müssen die Modelle vor der Codegenerierung miteinander in Verbindung gebracht werden. Dies wird als *verlinken* bezeichnet. Die Modellelemente des Ausgangsmodells werden dafür mit den Verlinkungen zu den Modellelementen des anderen Modells angereichert. Die Modelle, die miteinander verlinkt werden, müssen dabei nicht Instanzen desselben Metamodells sein. Abbildung 4-8 verdeutlicht das Modell-Weaving. Die Verlinkungen sind hierbei durch die gepunkteten Linien dargestellt. ([Sta07], Kap. 10.2)

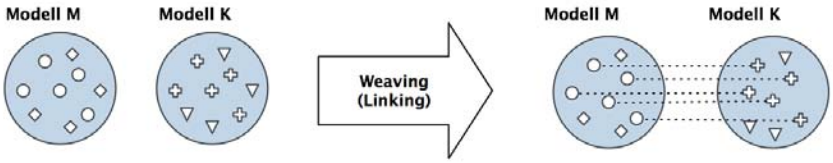


Abb. 4-8. Modell-Weaving, Quelle: ([Sta07], S. 200)

Bei der Modelltransformation wird im Gegensatz zur Modellmodifikation das Ausgangsmodell nicht verändert. Es besteht außerdem die Möglichkeit, mehrere Ausgangsmodelle zu einem Modell zusammenzuführen. Diese Form der Transformation wird beispielsweise angewandt, wenn das Ausgangsmodell ein anderes Metamodell hat als das Modell, in das es transformiert werden soll. Abbildung 4-9 zeigt diese Form der Modell-zu-Modell-Transformation. Die verschiedenen Metamodelle werden dabei durch die unterschiedlichen geometrischen Formen der Modellelemente der beiden Modelle dargestellt. ([Sta07], Kap. 10.2)

Für die Durchführung einer Transformation gibt es unterschiedliche Ansätze. Ähnlich der M2C-Transformation kann eine Transformation beispielsweise mittels einer Programmiersprache umgesetzt werden ([Pet06], Kap. 3.3). Diese Vorgehensweise ist jedoch insbesondere bei umfangreichen Transformationen sehr komplex ([Sta07], Kap. 10.3). Darüber hinaus gibt es daher spezielle Werkzeuge, die im Folgenden als *Transformationswerkzeuge* bezeichnet werden. Diese Werkzeuge nutzen eine *Transformationssprache* zur Beschreibung der Transformation ([Sta07], Kap. 10.3). Diese Transformationssprachen folgen in der Regel dem Konzept des

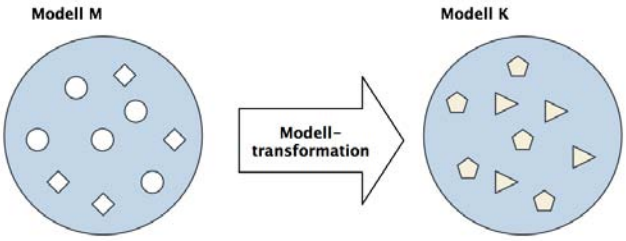


Abb. 4-9. Modelltransformation, Quelle: ([Sta07], S. 200)

Mappings. Bei einem Mapping werden *Mapping-Funktionen* definiert. Diese legen anhand von *Mapping-Regeln* fest, wie ein Modell transformiert wird. Die Mapping-Regeln werden hierfür auf Grund des Metamodells erstellt. Eine Mapping-Regel kann beispielsweise festlegen, dass jedes Modellelement, das Instanz eines speziellen Metamodellelements ist, durch die Transformation ein zusätzliches Attribut erhält. Eine Transformationssprache, die die Definition von Mappings ermöglicht, ist beispielsweise *Query, Views, and Transformations* (QVT). QVT ist der Standard der OMG für Modelltransformationen ([Mel04], Kap. 5).

Bei der Konzeption eines Transformationswerkzeugs müssen unterschiedliche Punkte durchdacht werden. Eine Schwierigkeit besteht beispielsweise im Umgang mit bidirektionalen Verweisen und Zyklen innerhalb eines Modells. Bei einem Zyklus verweist beispielsweise Element A auf Element B, Element B auf Element C und Element C auf Element A. Zwischen Element A und Element C können dabei beliebig viele Elemente liegen. Ein bidirektionaler Verweis ist ein Zyklus bei dem es nur zwei Elemente gibt. Bei einer Transformation können hierbei, für den Fall, dass die Transformation eines Elements erst abgeschlossen wird, wenn die Transformation seines Nachfolgers abgeschlossen wurde, Endlosschleifen entstehen. Transformationswerkzeuge müssen daher solche Eigenschaften eines Modells erkennen und entsprechend behandeln. ([Sta07], Kap. 10.3)

4.4.3. Cartridges

Aktuelle Generator-Frameworks erlauben das Modularisieren von Teilen des Generierungsprozesses zu sogenannten *Cartridges*. Der Begriff "Cartridge" wird dabei im Rahmen der modellgetriebenen Softwareentwicklung mit "Steckmodul" übersetzt. Ziel einer Cartridge ist es, die Erzeugung von speziellen Teilen des Zielsystems zusammenzufassen, um sie in verschiedenen Projekten wiederverwenden zu können. ([Sta07], Kap. 10.1)

Cartridges kapseln in der Regel spezielle Teile oder plattformspezifische Komponenten eines Softwaresystems. Beispielsweise kann die Schnittstelle zu einer Datenbank automatisch umgesetzt werden, indem beim Transformationsprozess eine Cartridge für ein *OR-Mapping*

Framework[1] verwendet wird. Bei der Generierung einer J2EE-Anwendung könnte eine Cartridge hingegen die Erzeugung der plattformspezifischen Teile des Zielsystems, wie Konfigurationsdateien, übernehmen. Durch die Verwendung von Cartridges wird ein Generator weniger komplex. Dies wird dadurch erreicht, dass der Generator, wie in Abbildung 4-10 dargestellt, anstelle von mehreren Aufrufen an die durch die Cartridge gekapselten Komponenten, nur einmal die Cartridge aufruft. ([Sta07], Kap. 8.1)

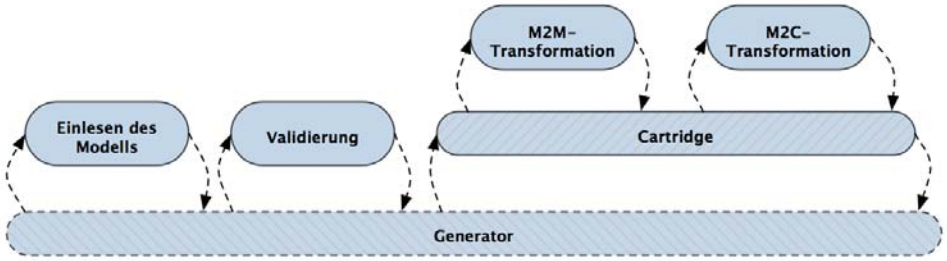


Abb. 4-10. Cartridges, Quelle: ([Sta07], S. 140)

4.4.4. Erneute Codegenerierung

Wichtiger Bestandteil der modellgetriebenen Softwareentwicklung ist das Zusammenspiel von Modellierung/Validierung und Generierung. Ähnlich wie bei der Modellierung und Validierung ist dies ein iterativer Prozess. Nach der ersten Codegenerierung werden typischerweise Änderungen am Modell vorgenommen. Diesen Änderungen folgt eine erneute Codegenerierung. ([Sta07], Kap. 2.2)

Die Codegenerierung ist kein spezielles Verfahren der modellgetriebenen Softwareentwicklung. Sie wird auch bei anderen Ansätzen der Softwareentwicklung angewandt. Beispiel hierfür ist die Codegenerierung aus UML-Modellen. UML-Modelle stellen die Klassen eines Programms und die Beziehungen der Klassen untereinander dar. Abbildung 4-11 zeigt, wie die Codegenerierung bei MDD innerhalb der gängigen Codegenerierungsverfahren einzuordnen ist.

Neben der Codegenerierung bei MDD gibt es das *Forward Engineering*, das *Reverse Engineering* und das *Roundtrip Engineering*. Bei der Codegenerierung aus UML-Modellen werden diese Codegenerierungsverfahren in der Regel angewandt. Beim Forward Engineering werden Änderungen am Modell vollzogen. Diese Änderungen werden dann automatisch in den Quelltext übertragen. Beim Reverse Engineering werden die Änderungen dagegen am Quelltext vorgenommen. Das Modell wird dabei automatisch an den Quelltext angepasst. Beim Forward Engineering und Reverse Engineering besteht folglich eine unidirektionale Beziehung

[1] Ein OR-Mapping (Object-Relational-Mapping) Framework ermöglicht dem Entwickler die automatische Persistierung von Objekten in einer Datenbank. Bei der Verwendung eines OR-Mapping Frameworks wird von der konkreten Datenbank abstrahiert. Der Entwickler nutzt die Funktionalität des Frameworks, anstatt direkt über einen Datenbanktreiber auf die Datenbank zuzugreifen. ([Bau06], Kap. 1.4)

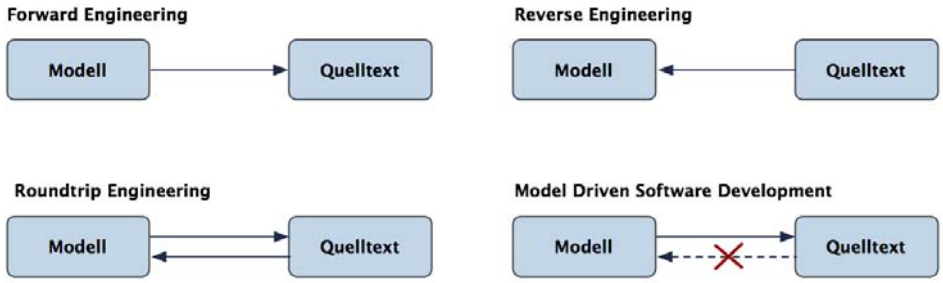


Abb. 4-11. Codegenerierungsverfahren, Quelle: ([Sta07], S. 45)

zwischen Modell und Quelltext. Das Roundtrip Engineering vereint Forward Engineering und Reverse Engineering. Änderungen können am Modell oder am Quelltext vorgenommen werden und der jeweils andere Teil wird entsprechend angepasst. Beim Roundtrip Engineering besteht somit eine bidirektionale Verbindung zwischen Modell und Quelltext. ([Sta07], Kap. 3.3)

Beim Forward Engineering, Reverse Engineering und Roundtrip Engineering haben Modell und Quelltext typischerweise dasselbe Abstraktionsniveau. Das bedeutet, dass das Modell genau das abbildet, was generiert wird. Sollen beispielsweise Klassen und deren Methoden generiert werden, sind diese komplett im Modell enthalten. Andernfalls wären das Reverse Engineering und damit auch das Roundtrip Engineering nicht möglich. Bei modellgetriebener Softwareentwicklung hingegen steht das Modell auf einer höheren Abstraktionsebene als der Quelltext. Modell und Code stehen daher nur in einer unidirektionalen Beziehung zueinander. Änderungen am Quelltext können nicht automatisch in das Modell aufgenommen werden. ([Sta07], Kap. 3.3)

Die erste Generierung von Quelltext aus einem Modell ist meistens unproblematisch. Es entstehen Quelltext-Dateien. Sehr häufig muss jedoch Code manuell hinzugefügt werden. Selten lässt sich eine Anwendung bei MDD zu 100% generieren. Der manuell hinzugefügte Code würde jedoch bei einer erneuten Generierung überschrieben werden. Um dieses Problem zu handhaben, existieren unterschiedliche Lösungsansätze. Zwei davon, die Verwendung von *Protected Regions* und die Trennung von generiertem und handgeschriebenem Code in unterschiedliche Dateien, werden im Folgenden erläutert. ([Pet06], Kap. 3.3)

Protected Regions sind geschützte Bereiche im Quelltext. Diese geschützten Bereiche werden bei einer erneuten Codegenerierung nicht mit generiertem Code überschrieben. Anfang und Ende einer Protected Region werden hierfür mit entsprechenden Kommentaren ausgezeichnet. Der folgende Pseudocode in Listing 4-3 zeigt das Schema einer Protected Region ([Tro07], Kap. 5.4).

```
1 public class Person {
2    ...
3    String name = null;
4
5    // anfang protected region
6       // hierhin kommt der manuell geschriebene Quelltext
7    // ende protected region
8    ...
9 }
```

Listing 4-3 Beispiel einer Protected Region, Quelle: ([Tro07], S. 177f.)

Protected Regions haben unterschiedliche Nachteile. Zum einen gibt es keine klare Trennung zwischen generiertem und manuell erstelltem Quelltext, da sich generierter und handgeschriebener Code innerhalb einer Datei vermischen. Des Weiteren kann der Entwickler nur innerhalb einer Protected Region Änderungen am Code vornehmen. Er kann nicht wie bei manuell erstelltem Code, alle Code-Teile beliebig ändern ([Pet06], Kap. 3.3). Ein weiteres Problem besteht darin, dass generierter Code, der manuell verändert wurde, unter Versionskontrolle gestellt werden muss. Dies ist erforderlich, um die Änderungen zu verwalten und sie, wenn das Entwicklerteam aus mehreren Personen besteht, jedem Entwickler zur Verfügung zu stellen. Generell sollte generierter Code jedoch nicht in einem *Versionskontrollsystem* erfasst werden. Dies hat den Grund, dass bei der modellgetriebenen Softwareentwicklung das Modell die Rolle des Quelltexts einnimmt. Die generierten Quelltexte sind nur eine Art Zwischenprodukt, ähnlich den Class-Dateien bei Java-Programmen ([Sta07], Kap. 2.2). Weiterhin wird der Generator bei der Verwendung von Protected Regions unter Umständen sehr komplex, da er die geschützten Bereiche verwalten, erkennen und erhalten muss. Schließlich kommt noch hinzu, dass der Entwickler bei Protected Regions den generierten Code zumindest zum Teil verstehen muss. Dies ist notwendig, um sinnvolle Änderungen und Erweiterungen am Quelltext vornehmen zu können. Generierter Code ist jedoch nicht immer einfach zu verstehen ([Sta07], Kap. 8.5). Die Vorteile von Protected Regions sind dagegen, dass der Generator die Kontrolle über die generierten Dateien behält. Wird beispielsweise ein Element im Modell gelöscht, kann der Generator die zugehörigen generierten Dateien ebenfalls löschen. Ein weiterer Vorteil ist, dass die vom Modell vorgegebene Architektur eingehalten wird ([Tro07], Kap. 5.4).

Für die Trennung von generiertem und manuell erstelltem Code in separate Dateien gibt es unterschiedliche Vorgehensweisen. Ein möglicher Ansatz ist die *dreistufige Vererbung*. Hierfür ist es erforderlich, dass der Quelltext, der generiert wird, in einer objekt-orientierten Programmiersprache verfasst ist. Des Weiteren wird vorausgesetzt, dass die Teile des zu generierenden Systems drei unterschiedliche Arten von Funktionalität besitzen. Diese sind bei Stahl et al. ([Sta07], S. 161) wie folgt definiert:

- Es gibt Funktionalität, die für alle Teile eines Typs gleich ist.

- Es gibt Funktionalität, die für jedes Teil unterschiedlich ist. Sie lässt sich jedoch aus den Informationen, die im Modell erfasst sind, generieren.

- Es gibt Funktionalität, die für jedes Teil unterschiedlich ist und vom Entwickler manuell implementiert werden muss.

Auf diesen Grundvoraussetzungen baut die dreistufige Vererbung auf. Zunächst wird dabei die Funktionalität implementiert, die für alle Teile eines Typs gleich ist. Diese Implementierung wird Teil der Plattform. Sie kann dem generierten Code beispielsweise über Programmbibliotheken zur Verfügung gestellt werden. Bei der Codegenerierung werden dann abstrakte Klassen erzeugt, die von den zuvor implementierten Klassen erben. Diese abstrakten Klassen setzen die Funktionalität um, die aus den Informationen im Modell generiert werden kann. Schließlich werden nicht-abstrakte Klassen manuell implementiert. Diese Klassen erben von den generierten, abstrakten Klassen und setzen die Funktionalität um, die nicht aus den Informationen des Modells generiert werden kann. Die manuell erstellten Klassen werden dann im System verwendet. ([Sta07], Kap. 8.5)

Neben der dreistufigen Vererbung gibt es weitere Ansätze zur Trennung von generiertem und manuell erstelltem Code. Diese basieren in der Regel auf *Entwurfsmustern* und werden daher von Stahl et al. als *Entwurfsmuster-basierte Integration* bezeichnet. Ein Entwurfsmuster, auch *Pattern* genannt, *«gibt eine bewährte, generische Lösung für ein immer wiederkehrendes Entwurfsproblem an, das in bestimmten Situationen auftritt»* ([Bal05], S. 74). Die Entwurfsmuster-basierte Integration legt dabei keine konkreten Vorgehensweisen fest, sondern umfasst unterschiedliche Konzepte. Diese Konzepte beschreiben, auf welche Art generierter und manuell implementierter Code zusammengebracht werden können. Der generierte Code kann zum Beispiel den manuell erstellten Code aufrufen oder umgekehrt. ([Sta07], Kap. 8.5)

Bei der Trennung von generiertem Code und manuell erstelltem Code in separate Dateien ergeben sich gegenüber der Verwendung von Protected Regions andere Probleme. Zunächst muss hierbei abhängig vom Typ der zu generierenden Dateien, beispielsweise Java-Dateien, Properties-Dateien oder XML-Dateien, eine andere Strategie entwickelt werden, um generierten Code und handgeschriebenen Code zusammenzuführen. Bei objektorientierten Sprachen kann beispielsweise mit Vererbung oder Delegation gearbeitet werden. Bei nicht objektorientierten Sprachen können *Includes* verwendet werden. Unter Includes werden hier Anweisungen verstanden, die den Code aus anderen Dateien ausführen ([Pet06], Kap. 3.3). Ein weiterer Nachteil ist, dass der Generator nicht die Kontrolle über alle Dateien behält. Bei strukturellen Änderungen beispielsweise werden die generierten Dateien vom Generator angepasst. Die manuell erstellten Dateien müssen jedoch per Hand geändert werden. Der Vorteil dieser Technik liegt hingegen darin, dass es eine klare Trennung von generiertem und handgeschriebenem Code gibt. Die generierten Artefakte müssen somit nicht in ein Versionskontrollsystem gestellt werden. ([Tro07], Kap. 5.4).

4.5. Interpreter

Neben dem Ansatz der Generierung von Quelltext aus Modellen gibt es einen weiteren Ansatz zur Verarbeitung von Modellen: sogenannte *Interpreter*. Ein Interpreter ist eine Software oder ein Teil einer Software, der Modelle zur Laufzeit auswertet. Der Interpreter führt dabei Aktionen aus, die durch das Modell beschrieben werden. Im Gegensatz dazu erzeugt ein Generator auf einem Modell basierend Quelltext, der später unabhängig vom Generator ausgeführt werden kann. Die Eigenschaften eines Interpreters sind zum großen Teil dieselben wie die eines Codegenerators. Beide Ansätze setzen beispielsweise ein formales Modell voraus und sind spezifisch für eine Plattform. ([Sta07], Kap. 9.1)

Das Konzept von Interpretern und Generatoren wird zum Beispiel bei Programmiersprachen angewandt. Das Modell wäre hierbei der Quelltext des Programms. Interpretierte Programmiersprachen, wie beispielsweise Ruby, werden direkt durch den Interpreter ausgeführt. Programmiersprachen wie C++ dagegen werden zunächst von einem *Compiler* in Maschinencode übersetzt, der dann unabhängig vom Compiler ausgeführt wird.

Interpreter und Generatoren sind keine sich gegenseitig ausschließenden Konzepte. Es gibt beispielsweise einen weiteren Ansatz zur Verarbeitung von Modellen, der eine Kombination von Generatoren und Interpretern ist. Ein Generator überführt hierbei die Modelle in ein Zwischenformat, das anschließend von einem Interpreter ausgeführt wird. Dieses Konzept wird ebenfalls bei Programmiersprachen wie zum Beispiel Java angewandt. Weitere Informationen zu Interpretern finden sich bei ([Sta07], Kap. 9). In dieser Arbeit wird auf dieses Thema nicht näher eingegangen, da Interpreter bei der Konzeption und Umsetzung des im Rahmen dieser Diplomarbeit zu entwickelnden Prototyps nicht angewandt wurden.

4.6. Model Driven Architecture

Model Driven Architecture (MDA) ist eine Spezifikation der *Object Management Group* (OMG) im Rahmen der modellgetriebenen Softwareentwicklung. Stahl et al. äußern sich im Bezug zur Einordnung der MDA im Kontext von MDD wie folgt.

> *«Wir sehen die MDA als die Standardisierungsinitiative der OMG zum Thema MDSD an. Da die MDA jedoch (bisher) nicht das gesamte MDSD-Spektrum abdeckt, kann man die MDA auch als eine MDSD-Ausprägung respektive Interpretation auffassen.»* ([Sta07], S. 377)

Die OMG wurde 1989 gegründet. Sie ist ein internationales Konsortium aus etwa 800 Firmen weltweit. Die OMG erstellt herstellerunabhängige Spezifikationen zur Interoperabilität und Portabilität von Softwaresystemen. Interoperabilität ist *«die Fähigkeit verschiedener Geräte oder Software, direkt miteinander zu kommunizieren.»* ([Bar03], S. 472) Unter Portabilität

wird hier die Modifizierung einer Anwendung verstanden, so dass sie in einer anderen Umgebung, zum Beispiel auf einem anderen Betriebssystem, in gleicher Weise arbeitet ([Bar03], S. 720). Die beiden primären Motivationen bei MDA sind daher Interoperabilität und Portabilität. Weitere bekannte Spezifikationen der OMG sind beispielsweise CORBA oder UML. ([Sta07], Anhang A.1)

Bei MDA wird die MOF als Metametamodell verwendet. Metamodelle sind somit Instanzen der MOF. Die MOF ist das Metamodell der UML, daher wird bei MDA oft die UML als Metamodell verwendet. Für den Austausch von Modellen zwischen verschiedenen MDA-Werkzeugen wird *XML Metadata Interchange* (XMI) eingesetzt. XMI ist ein Austauschformat für Modelle, die die MOF als Metametamodell verwenden, und durch die OMG standardisiert. ([Sta07], Anhang A.2)

Die MDA arbeitet mit drei zentralen Modellen: das *Platform Independent Model* (PIM), das *Platform Description Model* (PDM) und das *Platform Specific Model* (PSM). Diese dienen dazu, die Anwendungslogik unabhängig von der Plattform zu entwickeln. Das PIM beschreibt dabei die Anwendungslogik. Plattformspezifische Details werden hierfür bei diesem Modell weggelassen. Das PDM, auch *Platform Model* (PM) genannt, ist hingegen das Metamodell der Zielplattform. Es beschreibt die Struktur der Zielplattform. Das PDM für eine Datenbank enthält beispielsweise ein Element für Tabellen, welches aus Elementen für Spalten aufgebaut ist. Das PSM ist schließlich plattformspezifisch. Es enthält die mit zielplattformspezifischen Implementierungsdetails angereicherte Anwendungslogik. Das PSM kann beispielsweise Implementierungsdetails enthalten, die spezifisch für J2EE-Anwendungen sind. ([Sta07], Anhang A.2)

Bei MDA beginnt der Entwicklungsprozess mit der Modellierung des PIMs und des PDMs. Diese beiden Modelle dienen als Ausgangsmodelle für Modelltransformationen. Durch die Modelltransformation entsteht das PSM. Abbildung 4-12 stellt den MDA-Entwicklungsprozess schematisch dar. ([Pet06], Kap. 2.5)

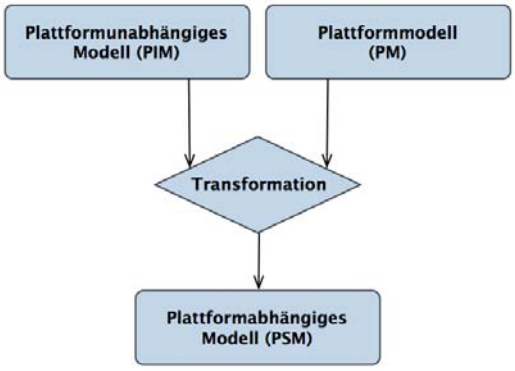


Abb. 4-12. Codegenerierungsverfahren, Quelle: ([Pet06], S. 107)

Aus dem PSM wird der Quelltext generiert. Vor der Codegenerierung können jedoch zuvor mehrmals Transformationen auf dem PSM ausgeführt werden. Bei jeder Transformation wird das Modell dabei mit Implementierungsdetails angereichert. Diese sind bei jeder Transformation spezifischer für die Zielplattform als bei der vorhergehenden Transformation. Das aus der ersten Transformation hervorgehende PSM kann beispielsweise spezifisch für J2EE sein. Es sagt jedoch noch nichts darüber aus, für welchen Applikationsserver es optimiert ist. Erst nach einer weiteren Transformation ist das PSM spezifisch für einen Applikationsserver (zum Beispiel für JBoss). ([Sta07], Anhang A.2)

In dieser Arbeit wird des Weiteren nicht näher auf MDA eingegangen, da dieser Standard für die Konzeption und Umsetzung des zu entwickelnden Prototyps nicht angewandt wird. Eine detaillierte Betrachtung von MDA findet sich bei Stahl et al. ([Sta07], Anhang A.2) und Petrasch et al. ([Pet06]).

Kapitel 5.

Umsetzungsmöglichkeiten

Für die Umsetzung von modellgetriebener Softwareentwicklung werden in der Regel unterschiedliche Werkzeuge in Kombination miteinander verwendet. Diese Werkzeuge decken dabei die folgenden Teilbereiche des modellgetriebenen Entwicklungsprozesses ab.

- Es muss ein geeignetes Metamodell definiert werden.
- Es wird ein Werkzeug zur Modellierung benötigt.
- Es ist ein Generator für die Transformationen erforderlich.

Im Folgenden werden für diese Teilbereiche unterschiedliche Werkzeuge vorgestellt. Die Werkzeuge decken dabei oftmals nicht nur einen Teilbereich ab. Manche Werkzeuge zur Modellierung legen beispielsweise die Technik zur Erstellung des Metamodells oder den zu verwendenden Generator fest. Ebenso muss für die Anwendung mancher Generatoren das Modell in einem speziellen Format vorliegen. Bei der Wahl der Werkzeuge und Techniken muss dies bedacht werden.

5.1. Metamodellierung

Die Metamodellierung ist ein wesentlicher Teil der Entwicklung einer DSL. Es gibt unterschiedliche Metametamodelle, die hierfür verwendet werden können. Neben der MOF, die bei der MDA angewandt wird, gibt es beispielsweise die folgenden drei Möglichkeiten ein Metamodell zu definieren:

- die Verwendung des Metametamodells *Ecore*,
- die Erstellung eines Metamodells mittels Klassen einer Programmiersprache,
- die Beschreibung des Metamodells mittels XML.

Im Folgenden werden diese vier Techniken der Metamodellierung beschrieben.

5.1.1. MOF und UML

Die OMG definiert 4 (Meta-) Ebenen zur Beschreibung eines Systems. Die Ebenen sind von M0 bis M3 durchnummeriert, wie in Abbildung 5-1 dargestellt. Die MOF ist dabei das Metametamodell und befindet sich auf Ebene M3. Die MOF beschreibt sich selbst. Anhang A.2 zeigt einen vereinfachten Ausschnitt der MOF. Auf Ebene M2 befindet sich die UML oder ein anders Metamodell, das eine Instanz der MOF ist. Die hier betrachtete Version der UML ist die UML 2.0. Im Folgenden wird der Begriff "UML" ohne Versionsbezeichnung verwendet. Die Aussagen zur UML beziehen sich jedoch auf diese Version. ([Sta07], Kap. 5.2)

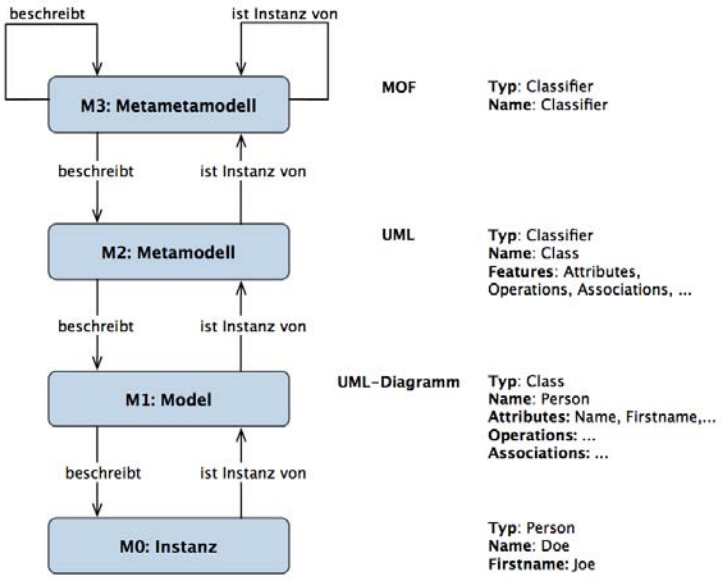


Abb. 5-1. Metaebenen, Quelle: ([Sta07], S. 62)

Bei der Verwendung der MOF zur Metamodellierung kann die UML als Metamodell verwendet werden. Dies hat den Vorteil, dass die UML zum einen durch die OMG standardisiert ist. Zum anderen gibt es unterschiedliche Werkzeuge zur Erstellung und Bearbeitung von UML-Modellen. Bei der Bearbeitung von Modellen, die die UML als Metamodell verwenden, ist der Entwickler daher nicht auf ein spezifisches Werkzeug festgelegt. Des Weiteren können UML-Modelle von unterschiedlichen Systemen verarbeitet werden. Neben der Verwendung der UML als Metamodell besteht die Möglichkeit, mittels der MOF ein Metamodell, das spezifisch für eine Domäne ist, zu entwickeln. Hierbei gehen jedoch die Vorteile der UML, wie zum Beispiel die Werkzeugunterstützung zur Modellierung, verloren. In den meisten Fällen ist daher ein anderes Metametamodell für diesen Zweck besser geeignet. ([Sta07], Kap. 5.2)

«Die UML ist ein allgemeines Metamodell [...]*»* ([Sta07], S. 65), das der Beschreibung von Softwaresystemen dient. Bei der modellgetriebenen Softwareentwicklung wird jedoch in der Regel ein Metamodell benötigt, das die speziellen Eigenschaften einer Domäne wiedergibt.

Zu diesem Zweck kann die UML erweitert werden. Diese Erweiterung kann durch die Definition von neuen UML-Elementen oder durch die Erstellung von *UML-Profilen* vorgenommen werden. ([Sta07], Kap. 5.2)

Bei der Erweiterung der UML um neue Elemente erben die neuen Elemente von bereits existierenden UML-Elementen. Die neuen Elemente sind somit Instanzen von MOF-Elementen. Abbildung 5-2 verdeutlicht diesen Ansatz. ([Sta07], Kap. 5.2)

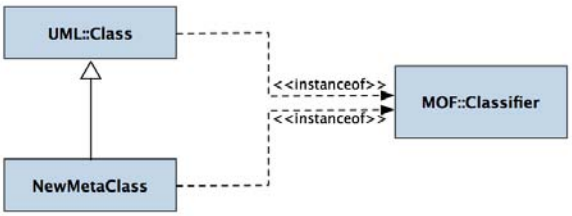


Abb. 5-2. Erweiterung der UML durch neue Metamodellelemente, Quelle: ([Sta07], S. 66)

Bei der Erstellung eines Modells unter Verwendung der so erweiterten UML können von den neuen UML-Elementen Instanzen erstellt werden. Die Darstellung der Instanzen hängt von dem gewählten UML-Werkzeug ab. Abbildung 5-3 zeigt mögliche Darstellungsarten. In Version (a) in Abbildung 5-3 wird die Instanzbeziehung zwischen `MetaClass1` und `NewMetaClass` in die Darstellung des neuen UML-Elements aufgenommen. Häufig wird das neue UML-Element auch als *Stereotyp* notiert, wie in Version (b) in Abbildung 5-3 dargestellt ([Sta07], Kap. 5.2). Ein Stereotyp ist ein UML-Element, das dem Modellelement, auf das es angewandt wird, zusätzliche Eigenschaften hinzufügt ([OMG05], Kap. 4.6). In Version (c) wird das UML-Element als *Tagged Value* notiert. Ein Tagged Value ist ein Schlüssel-Werte-Paar, das zu jedem Modellelement hinzugefügt werden kann. Es wird in geschweiften Klammern notiert, beispielsweise `{metaclass = newMetaClass}` ([OMG05], Kap. 5.17). Version (d) zeigt die Instanz des neuen UML-Elements in einer eigenen graphischen Notation.

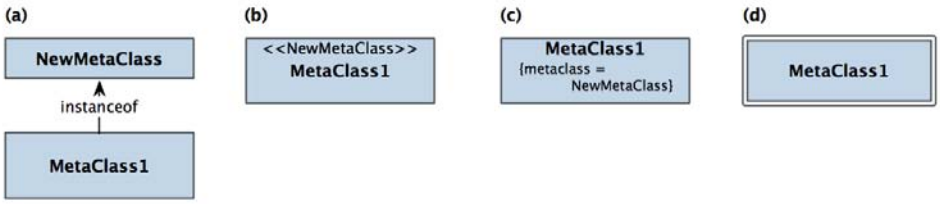


Abb. 5-3. Darstellung eines neuen UML-Elements, Quelle: ([Sta07], S. 67)

Die zweite Möglichkeit zur Erweiterung der UML sind UML-Profile, im Folgenden als *Profil* bezeichnet. In einem UML-Profil sind UML-Erweiterungen zusammengefasst, mit dem Ziel, die UML an eine fachliche oder technische Domäne anzupassen. Ein Profil wird als ein UML-Paket

mit dem Stereotyp «profile» dargestellt (siehe Abbildung 5-4, Profildefinition). ([Mil06], Anhang B)

Ein UML-Profil enthält Stereotypen, *Tag Definitions* und Constraints. Tag Definitions sind mögliche Eigenschaften eines Modellelements. Eine Tag Definition, die für ein Modellelement angegeben ist, wird über ein Tagged Value dargestellt. Ein Stereotyp definiert zum einen mittels der Tag Definitions Eigenschaften und zum anderen Constraints. Die Modellelemente, die mit dem Stereotyp versehen sind, erhalten die Eigenschaften und Constraints des Stereotyps, wie in Abbildung 5-4 im Teil "Profilanwendung" gezeigt ([OMG05], Kap. 4.6). Der Typ der Modellelemente, auf die ein Stereotyp angewendet werden kann, wird bei einer Profildefinition über eine *Extension* festgelegt. Eine Extension ist ein Pfeil, dessen Pfeilspitze ausgefüllt ist. Sie verbindet den Stereotyp mit dem entsprechenden Metamodellelement, wie beispielsweise dem MOF-Element `Class`. Ein Metamodellelement wird hierbei mit dem Stereotyp «metaclass» gekennzeichnet ([Mil06], Kap. B.5). Abbildung 5-4 zeigt die Definition und Anwendung eines Profils, wobei eine Extension verwendet wird.

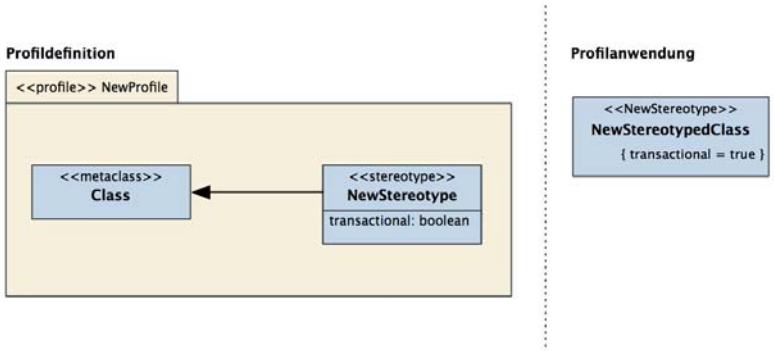


Abb. 5-4. Definition und Anwendung eines UML-Profils, Quelle: angelehnt an ([Sta07], S. 69)

Wichtiger Bestandteil eines Metamodells sind Constraints. Sie sind erforderlich, um Modelle, die Instanzen dieses Metamodells sind, validieren zu können. Bei der Verwendung von UML können Constraints mit einer beliebigen Sprache definiert werden. Die OMG hat zur Formulierung von Constraints die *Object Constraint Language* (OCL) standardisiert. OCL wurde von IBM entwickelt und ist Teil der UML-Spezifikation ([Mil06], Anhang A). Ein Profil kann neue Constraints definieren. Es kann die bestehenden Constraints der UML jedoch nicht aufheben ([Sta07], Kap. 5.2).

Zusammenfassend ist zur UML zu sagen, dass sie unterschiedliche Vorteile mit sich bringt. Dazu gehören die Auswahl an unterschiedlichen Modellierungswerkzeugen und die Standardisierung durch die OMG. Der Nachteil der UML ist jedoch, dass sie zwar erweitert, aber nicht reduziert werden kann. UML-Werkzeuge stellen daher immer alle Modellierungsmöglichkeiten zur Verfügung. Dies kann zu einem Überangebot an Funktionalität führen, das unter Umständen nicht erwünscht ist. ([Sta07], Kap. 5.2)

5.1.2. Ecore und EMF

Das *Eclipse Modeling Framework* (EMF) wird im Rahmen des *Eclipse Modeling Projects*[1] der *Eclipse Foundation* entwickelt. Die Eclipse Foundation ist eine gemeinnützige Organisation, die die unterschiedlichen Projekte der Eclipse-Open-Source-Gemeinschaft leitet. Diese Projekte konzentrieren sich auf die Entwicklung und Erweiterung einer Open-Source-IDE, genannt *Eclipse*, deren aktuelle Version zur Zeit der Anfertigung dieser Diplomarbeit *Eclipse Europa* ist. Die Funktionalität von Eclipse kann durch Komponenten, sogenannte *Plugins*, erweitert werden. ([Ecl08a])

EMF ist ein solches Plugin für Eclipse. Das Plugin stellt ein Java Framework bereit, das der Erzeugung von Modellierungswerkzeugen dient. Diese Modellierungswerkzeuge sind spezifisch für ein Metamodell. Das Metamodell kann dabei mit EMF definiert werden. EMF folgt dem Ansatz der modellgetriebenen Softwareentwicklung. Ein Modellierungswerkzeug wird dazu aus dem Metamodell und einem weiteren Modell, das implementierungsspezifische Details enthält, automatisch generiert. Mittels dieser generierten Modellierungswerkzeuge können Modelle bearbeitet und persistiert werden. ([Ecl05b])

EMF verwendet Ecore zur Metamodellierung. Ecore hat Ähnlichkeit mit der *Essential MOF* (EMOF), die eine reduzierte Version der MOF ist ([Sta07], Kap. 5.2). Die Elemente der EMOF und ihre Hierarchie sind in Anhang A.3 abgebildet. Ebenso wie die MOF beschreibt sich Ecore selbst ([Ste03], Kap. 2.3). Im Gegensatz zur MOF wird jedoch bei der Verwendung von Ecore das gesamte Metamodell definiert. Ein solches Metamodell wird dabei mit der Dateiendung ".ecore" im XMI-Format, einem XML-Format, gespeichert. ([Sta07], Kap. 5.2)

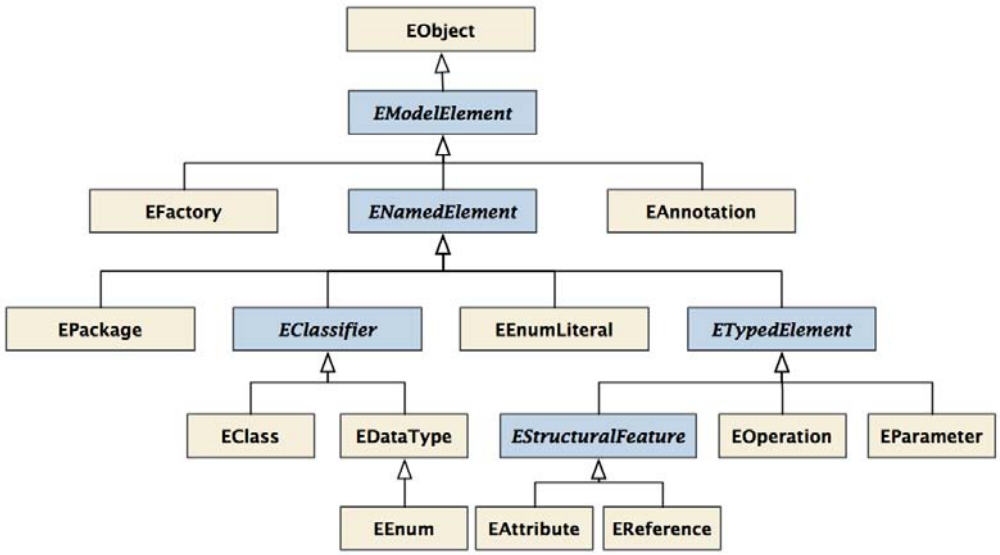


Abb. 5-5. Metametamodell Ecore, Quelle: ([Ecl05b])

[1] Mehr Informationen zum Eclipse Modeling Project sind zu finden unter http://www.eclipse.org/modeling/.

Abbildung 5-5 zeigt die Klassenhierarchie des Metametamodells Ecore. Abstrakte Klassen sind in der Abbildung blau gefärbt. Bei der Erstellung eines Metamodells mittels Ecore werden Instanzen der in der Abbildung gezeigten Elemente erstellt. Die Modellelemente eines Metamodells sind beispielsweise Instanzen der Unterklassen der abstrakten Klasse `EClassifer`.

Für die Erstellung eines Metamodells mittels EMF gibt es mehrere Möglichkeiten. Es kann als Ecore-Modell, als XML Schema, als UML-Diagramm oder als annotierte Java-Interfaces definiert werden. Aus dem Format, welches zur Definition verwendet wird, können die jeweils anderen Formate dann automatisch erzeugt werden. Dieses Prinzip ist in Abbildung 5-6 schematisch dargestellt. ([Ste03], Kap. 2.4)

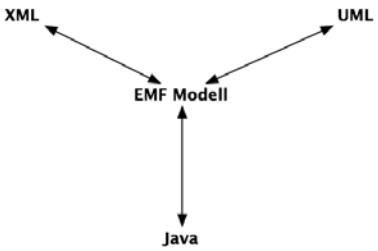


Abb. 5-6. EMF, Quelle: ([Ste03], Kap. 2.1)

«EMF consists of two fundamental frameworks: the core framework and EMF.Edit.» ([Ecl05b]) Das "Kern-Framework" bietet grundlegende Unterstützung zur Generierung der Java-Klassen für ein Ecore-Modell. Hierfür wird der Codegenerator *Java Emitter Templates* (JET) verwendet ([Ste03], Kap. 11.1). EMF.Edit erweitert das "Kern-Framework" hauptsächlich um die folgenden Bestandteile: ([Ecl05b])

- Mittels EMF.Edit werden Adapter für die Klassen, die die Elemente des Metamodells repräsentieren, erstellt. Diese Adapter folgen dabei nicht dem klassischen *Adapter*-Pattern, wie es bei Erich Gamma definiert wird.[2] Stattdessen erweitern sie die Funktionalität einer Klasse. Sie dienen nicht, wie es das Adapter-Pattern bei Gamma vorsieht, der Übersetzung einer Schnittstelle in eine andere. Adapter bei Eclipse, wozu auch die generierten Adapter gehören, folgen dem *Extension Object*-Pattern[3]. ([Gam03], Kap. 31)

  Die Adapter ermöglichen beispielsweise die Umsetzung des *Command*-Patterns. Bei der Verwendung des Command-Patterns werden Aktionen in Objekten gekapselt.[4] Dadurch können unter anderem Änderungen an einem Modell mit Hilfe einer Undo-Funktion rückgängig gemacht werden.

- Mit Hilfe des EMF.Edit-Frameworks kann ein einfacher Editor zum Bearbeiten der Modelle erstellt werden. Der Editor stellt die Modelle in einer Baumansicht dar.

5.1.3. Metamodellierung mittels Klassen

Bei der Metamodellierung mittels Klassen wird das Metamodell in einer beliebigen Programmiersprache implementiert. Hierfür wird für jedes Metamodellelement eine Klasse entwickelt. Modelle, die Instanzen eines solchen Metamodells sind, bestehen dann aus Objekten, die sich

[2] Vgl. hierzu das Entwurfsmuster "Adapter" bei Gamma et al. (([Gam94], S. 139)).
[3] Siehe hierzu ([Gam03], Kap. 31)
[4] Vgl. hierzu das Entwurfsmuster "Command" bei Gamma et al. (([Gam94], S. 233))

gegenseitig referenzieren. Die Informationen, die in einem Modell enthalten sind, sind in den Attributen der Objekte gespeichert. Das Metametamodell ist bei dieser Art des Metamodells die zugrunde liegende Programmiersprache. Zur Vereinfachung der Verarbeitung solcher Metamodelle, wird die verwendete Programmiersprache in der Regel durch geeignete Konventionen eingeschränkt. Eine Einschränkung der Programmiersprache Java könnte beispielsweise sein, dass jede Klasse einen `public`-Konstruktor haben muss. ([Sta07], Kap. 5.2)

Der Vorteil bei dieser Art der Metamodellierung ist die leichte Erweiterbarkeit der Funktionalität eines Metamodells. Benötigte Funktionen, wie beispielsweise die automatische Berechnung eines Werts A aus einem Wert B eines Modellelements, können dem Metamodell über das Erstellen einer geeigneten Methode in der entsprechenden Klasse hinzugefügt werden. Ein weiterer Vorteil ist, dass die Entwicklung des Metamodells durch die Verwendung einer leistungsstarken IDE unterstützt werden kann. ([Sta07], Kap. 5.2)

Diese Art der Repräsentation eines Metamodells wird oft zusätzlich zu einer der anderen, hier vorgestellten Arten genutzt. EMF beispielsweise bietet die Möglichkeit aus einem Ecore-Modell automatisch den Elementen entsprechende Java-Klassen zu generieren. Die meisten UML-Werkzeuge verwenden ebenfalls Klassen beziehungsweise von diesen erstellte Objekte für die interne Repräsentation eines Modells. Hierbei ist die Abstraktion bezüglich der Metaebenen eines Modells nicht mehr eindeutig. Solche Klassen repräsentieren abstrakt betrachtet ein UML-Metamodell beziehungsweise ein Ecore-Modell. Somit ist die MOF beziehungsweise Ecore das Metametamodell. Aus technischer Sicht sind diese Modelle jedoch als Instanzen einer Programmiersprache aufzufassen. Das Metametamodell ist aus dieser Sicht die jeweilige Programmiersprache. ([Sta07], Kap. 5.2)

5.1.4. XML/XSD

Die vierte hier vorgestellte Möglichkeit der Metamodellierung ist die Metamodellierung mit XML. Hierbei wird das Metamodell mittels einer *XML Schema Definition* (XSD) erstellt. Eine XSD ist ein XML-Dokument, das eine Struktur für XML-Dokumente definiert. Sie legt unterschiedliche Regeln fest, die ein XML-Dokument, welches dem Schema folgt, einhalten muss, damit es valide ist. Das XML-Dokument ist hierbei das Modell. Eine XSD folgt selbst einer XSD. XSD ist somit auch das Metametamodell. Der Vorteil bei der Verwendung von XML ist, dass es mehrere Werkzeuge und Programmbibliotheken zum editieren und parsen von XML-Dokumenten gibt.[5] ([Sta07], Kap. 5.2)

[5] Mehr Informationen zur Erstellung und Anwendung von XSD findet sich unter http://www.w3schools.com/schema/default.asp.

5.2. Modellierung

Ein weiterer wesentlicher Bestandteil der modellgetriebenen Softwareentwicklung ist die Modellierung. Dafür werden Modellierungswerkzeuge benötigt. Diese basieren auf einem zuvor definierten Metamodell. Sie umfassen im Wesentlichen eine graphische Benutzeroberfläche und die konkrete Syntax der DSL.

Für die Modellierung gibt es unterschiedliche Möglichkeiten. Zunächst kann die UML oder XML dazu verwendet werden. Die Modellierung mit UML richtet sich dabei nach den Erweiterungen der UML im Metamodell. Das bedeutet zum einen, dass für das Metamodell definierte UML-Profile bei der Modellierung angewandt werden (vgl. Abbildung 5-4). Zum anderen werden, wenn das Metamodell neu definierte UML-Elemente enthält, bei der Modellierung Instanzen von diesen erstellt (vgl. Abbildung 5-3). Eine ausführliche Beschreibung der Verwendung der UML findet sich in ([Mil06]). Bei der Modellierung mit XML richtet sich die Erstellung des XML-Dokuments nach der zuvor definierten XSD.

Eine weitere Möglichkeit für die Modellierung sind die mit EMF automatisch generierbaren Modellierungswerkzeuge. EMF wurde in Abschnitt 5.1.2 bereits vorgestellt. An dieser Stelle wird daher dieses Framework nicht eingehender beschrieben. Darüber hinaus gibt es jedoch weitere Technologien, die speziell für die Erstellung von Modellen oder die Erzeugung von Modellierungswerkzeugen konzipiert sind. Diese werden im Folgenden beschrieben.

5.2.1. GMF

Das *Graphical Modeling Framework* (GMF) ist ein Plugin für Eclipse. Es dient der Entwicklung von graphischen Editoren zur Erstellung und Persistierung von Modellen. GMF verbindet das *Eclipse Modeling Framework* (EMF) und das *Graphical Editing Framework* (GEF) ([Ecl05a]). Hierbei ist GEF ein Framework zur graphischen Erstellung und Darstellung von Modellen ([Ecl07b]). Bei GMF wird GEF für die Entwicklung der graphischen Editoren verwendet. Mittels EMF werden die Modelle gespeichert und verwaltet ([Ecl05a]). Abbildung 5-7 zeigt schematisch den Aufbau von GMF.

Ein mit GMF erstellter Editor zur Modellierung wird im Folgenden als *GMF-Modellierungstool* bezeichnet. Ein GMF-Modellierungstool benötigt zwei Arten von Informationen zu einem Modell. Es benötigt semantische Informationen und graphische Informationen. Die semantischen Informationen sind das Modell selbst. Sie geben beispielsweise an, welche Elemente im Modell enthalten sind und wie sie miteinander in Verbindung stehen. Die graphischen Informationen werden für die Darstellung des Modells verwendet. Sie beschreiben zum Beispiel, wie die Elemente des Modells konkret angezeigt werden. Das Modell wird im XMI-Format gespeichert. Die graphischen Informationen werden ebenfalls getrennt von dem Modell im XMI-Format gespeichert. Diese Trennung von semantischen und graphischen Informationen ermöglicht es, ein Modell auf unterschiedliche Weise ohne erneute Modellierung darzustellen.

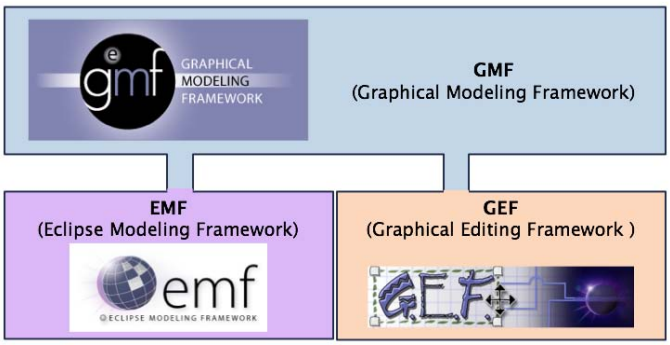


Abb. 5-7. Aufbau von GMF

Für eine andere Darstellung des Modells werden lediglich andere graphische Informationen benötigt. ([Ecl05a])

5.2.2. Xtext

Xtext ist ein Framework zur Definition von textuellen DSLs. Es ist Teil des *openArchitectureWare*-Projekts.[6] Xtext ermöglicht die Beschreibung der konkreten und abstrakten Syntax einer DSL durch eine vereinfachte EBNF (Extended Backus-Naur-Form). Die EBNF ist eine *«Notationsform* [...] *zur Definition der Syntax einer Programmiersprache»* ([Bar03], S. 91).

Listing 5-1 zeigt eine Xtext-Grammatik. Zunächst wird in der Grammatik in Zeile 1 festgelegt, dass ein Modell aus einem oder mehreren Elementen, in der Grammatik als `Entity` bezeichnet, besteht. Diese Festlegung bezieht sich auf die abstrakte Syntax der DSL. Die Zeilen 2 bis 4 legen fest, wie ein Element beschrieben wird. Dies sind Angaben zur konkreten Syntax der DSL. Des Weiteren wird in Zeile 3 im Rahmen der abstrakten Syntax definiert, dass ein Element beliebig viele Attribute besitzen kann. Zeile 5 sagt über die konkrete Syntax schließlich aus, in welcher Weise ein Attribut angegeben wird. ([Sta07], Kap. 6.2)

```
1 Model     :  (entities+=Entity)+;
2 Entity    :  "Entity" name=ID "{"
3                 (attributes+=Attribute)*
4              "}";
5 Attribute:  type=ID name=ID;
```

Listing 5-1 Xtext-Grammatik, Quelle: ([Sta07], S. 104)

Die angeführte Xtext-Grammatik beschreibt die in Listing 5-2 folgende DSL. Diese DSL legt fest, dass es zwei Metamodellelemente gibt: `Kunde` und `Adresse`. Des Weiteren kann ein Modellelement des Typs `Kunde` auf ein Modellelement des Typs `Adresse` verweisen.

[6] Informationen zum openArchitectureWare-Projekt finden sich unter ([ope08]).

```
1  Entity Kunde {
2     String name
3     Adresse adresse
4  }
5
6  Entity Adresse {
7     String strasse
8     String plz
9     String ort
10 }
```

Listing 5-2 DSL in Xtext, Quelle: ([Sta07], S. 104)

Aus der Xtext-Grammatik können ein Parser, das Metamodell und ein Eclipse-basierter Editor zur Modellierung generiert werden. Der generierte Editor hat dabei DSL-spezifische Eigenschaften wie Syntax-Highlighting oder eine automatische Codevervollständigung. Das Metamodell wird als Ecore-Modell generiert. Xtext legt somit fest, dass Ecore als Metametamodell verwendet wird. ([Sta07], Kap. 6.2)

5.2.3. Weitere Modellierungswerkzeuge

Neben GMF und Xtext gibt es eine Reihe von weiteren Modellierungswerkzeugen. Drei dieser Werkzeuge werden im Folgenden aufgeführt. Sie werden jedoch nicht ausführlich beschrieben, da sie entweder nur kommerziell nutzbar oder auf eine spezielle Ausprägung von modellgetriebener Softwareentwicklung spezialisiert sind.

Ein kommerzielles Programm zur Modellierung und Generierung von Software ist *ArcStyler* von der Firma Interactive Objects[7]. Dieses Programm ist speziell für den Entwicklungsprozess im Rahmen der MDA konzipiert. ArcStyler ist in Java implementiert. Es ist hauptsächlich auf die Erstellung von Software für J2EE oder .NET spezialisiert. Für die Modellierung wird dabei UML verwendet. ([Int08])

Ein weiteres kommerzielles Programm ist *Enterprise Architect* von der Firma Sparx Systems[8]. Enterprise Architect ist ein UML-Modellierungswerkzeug, das ebenfalls Unterstützung für den MDA-Entwicklungsprozess bietet. Das Programm ermöglicht dafür Template-basierte Transformationen. Diese dienen, wie es die MDA vorsieht, der Umwandlung von PIMs in PSMs und PSMs in Quelltext. ([Spa07])

Das *Generic Modeling Environment* (GME) ist ein Programm zur Erstellung von domänenspezifischen Modellierungstools für das Betriebssystem Windows. GME ist ein Open-Source-Werkzeug, das am Institut für Softwareintegrierte Systeme (ISIS) der Universität

[7] Siehe http://www.interactive-objects.com/
[8] Siehe http://www.sparxsystems.com/

von Vanderbilt[9] entwickelt wurde. GME wurde dabei ursprünglich im Rahmen von *Model-Integrated Computing* (MIC) entwickelt und eingesetzt.[10] ([Sta07], Kap. 6.2)

MIC ist eine spezielle Ausprägung der modellgetriebenen Softwareentwicklung. Es wurde im Umfeld von Echtzeitsystemen und eingebetteten Systemen entwickelt. Bei MIC wird ein System durch mehrere Modelle abgebildet. Die Modelle beschreiben verschiedene Teile des Systems. An das System werden hohe Anforderungen in Bezug auf Zuverlässigkeit gestellt. Die Validierung der Modelle ist daher ein zentraler Bestandteil bei diesem Ansatz.[11] ([Sta07], Kap. 3.2)

5.3. Generatoren

Den dritten Teilbereich der modellgetriebenen Softwareentwicklung bilden die Generatoren. Hierfür gibt es eine Reihe von Open-Source- und kommerziellen Generatoren. Open-Source-Generatoren sind beispielsweise *JET*, *AndroMDA*, *openArchitecureWare* und *Mod Transf.* Kommerzielle Generatoren sind zum Beispiel *BITPlan smartGENERATOR* oder *Codagen Architect*. Im Folgenden werden drei Open-Source-Generatoren vorgestellt.

5.3.1. JET

Java Emitter Templates (JET) ist ein Codegenerator, der im Rahmen des Eclipse Projektes *Model To Text* (M2T) entwickelt wird. M2T ist ein Projekt, das sich der Generierung von Textdokumenten aus Modellen widmet. Hierfür werden unter anderem Werkzeuge für die M2C-Transformation entwickelt, die sich nach Industriestandards, wie die der OMG, richten. Die derzeit aktuelle Version von JET wird auch als *JET2* bezeichnet. Die folgenden Aussagen beziehen sich auf diese Version, wobei der Begriff "JET" synonym zu JET2 verwendet wird. ([Ecl08e])

JET folgt bei der Codegenerierung einer Kombination von zwei Ansätzen: dem Template-basierten Ansatz und dem Ansatz der Generierung von Code mittels einer Programmiersprache. Der Entwickler erstellt dabei zunächst Templates. Aus diesen Templates werden dann durch JET Java-Klassen erzeugt. Diese Java-Klassen dienen der Erzeugung des zu generierenden Quelltextes. In Anhang A.4.1 befindet sich ein JET-Template und die dafür generierte Java-Klasse. ([Pop04])

Ein Modell muss, um von JET verarbeitet werden zu können, im XML-Format vorliegen. Dies schließt auch EMF-Modelle ein. Bei der Verarbeitung kann dann aus dem Modell beliebiger Text erzeugt werden. JET verwendet dabei, auf Grund des XML-Formats der Modelle, XPath, um Informationen aus einem Modell zu erhalten. ([Gov07])

[9] Siehe http://www.isis.vanderbilt.edu/
[10] Mehr Informationen zu GME siehe http://www.isis.vanderbilt.edu/projects/gme/index.html
[11] Mehr Informationen zu MIC siehe http://www.isis.vanderbilt.edu/research/mic.html

Ein JET-Template besteht aus statischem Text und Tags. Es gibt vier Arten von JET-Tags: *Control-Tags*, *Format-Tags*, *Java-Tags* und *Workspace-Tags*. Control-Tags dienen dem Zugriff auf die Informationen des Ausgangsmodells und kontrollieren die Ausführung des Templates. Das Tag `c:get` beispielsweise ist ein Control-Tag. Dieses Tag liest aus dem Ausgangsmodell einen Wert und schreibt diesen in den generierten Text. In Zeile 1 von Listing 5-3 wird beispielsweise ein `c:get`-Tag verwendet. Format-Tags wandeln Text nach vorgegebenen Regeln um. Das Format-Tag `f:formatNow` beispielsweise formatiert das aktuelle Datum und schreibt es in den generierten Text. Java-Tags sind spezielle Tags für die Generierung von Java-Quelltext. Workspace-Tags werden verwendet um Dateien zu erzeugen. Das Workspace-Tag `ws:copyFile` beispielsweise kopiert eine Datei. ([Ecl06])

Listing 5-3 zeigt ein JET-Template, das eine Java-Klasse mit einer `main`-Methode erzeugt. Hierbei wird über das Tag `c:get` zunächst in Zeile 1 der Klassenname aus dem Modell ausgelesen. Der Ausdruck `/class/@name` greift dafür auf das Attribut `name` des XML-Elements `class` zu. Dieses Attribut enthält den Klassennamen. Im Anschluss wird der String, der durch die erzeugte Methode ausgegeben wird, in Zeile 3 ausgelesen. Dafür wird über den Ausdruck `/class/phrase` auf den Inhalt des XML-Elements `phrase` zugegriffen. Dieses Element muss sich unterhalb des XML-Elements `class` befinden. In Anhang A.4.2 befindet sich ein Beispielmodell, das mit diesem Template verarbeitet werden kann und ein weiteres JET-Template, welches das Modell in eine Ruby-Klasse umwandelt.

```
1 public class <c:get select="/class/@name" /> {
2    public static void main(String[] args) {
3       System.out.println("<c:get select="/class/phrase" />");
4    }
5 }
```

Listing 5-3 Beispiel eines JET-Templates zur Erzeugung einer Java-Klasse, Quelle: ([Gov07])

JET nutzt für die Zusammenführung von handgeschriebenem und generiertem Text Protected Regions. Protected Regions werden mit dem Control-Tag `c:userRegion` ausgezeichnet. Für die Erzeugung von initialem Text wird dabei das Tag `c:initialCode` verwendet, wie in Listing 5-4 dargestellt. Hierbei ist zu beachten, dass der gesamte Text zwischen den Tags `<c:userRegion>` und `<c:initialCode>` beziehungsweise den entsprechenden schließenden Tags als Auszeichnung für die Protected Regions verwendet wird. Das bedeutet, dass in den generierten Quelltexten der Code, der zwischen `//BEGIN user code` und `//END user code` steht, nicht vom Codegenerator überschrieben wird. ([Ecl08b])

```
1 <c:userRegion>//BEGIN user code<c:initialCode>
2 //todo insert your code here
3 </c:initialCode>//END user code</c:userRegion>
```

Listing 5-4 Auszeichnung einer Protected Region in einem JET-Template, Quelle: ([Ecl08b])

Speziell für die Generierung von Java-Quelltext kann *JMerge* für die erneute Codegenerierung verwendet werden. JMerge ist ein Werkzeug zum Zusammenführen von handgeschriebenem und generiertem Java-Quelltext. Es wurde im Rahmen von EMF entwickelt ([Ecl08d]). JMerge ermöglicht es, Java-Elemente wie Klassen, Methoden oder Attribute als Protected Region auszuzeichnen. Die Auszeichnung erfolgt dabei über das Hinzufügen der Annotation `@generated` in einem JavaDoc-Kommentar ([Ecl08c]).

5.3.2. openArchitectureWare

openArchitectureWare (oAW) ist ein MDD/MDA-Generator-Framework, das in Java implementiert ist. Es ermöglicht das Validieren und Transformieren von Modellen. Darüber hinaus erlaubt oAW die Verwendung von unterschiedlichen Modellierungswerkzeugen zur Erzeugung des zu transformierenden Modells, wie beispielsweise EMF, GMF oder ein beliebiges UML-Werkzeug. oAW kann dafür unter anderem EMF-Modelle, Modelle im XML-Format und UML-Modelle verarbeiten. ([ope08])

Für eine Transformation mit oAW werden *Workflows* verwendet. Ein Workflow kontrolliert den Ablauf einer Transformation. Er legt dafür die Abschnitte eines Transformationsdurchlaufes fest, wie beispielsweise das Laden des Modells, die Modellvalidierung und die Transformation beziehungsweise Generierung. Für die Definition eines Workflows werden eine XML-Datei und eine Properties-Datei erstellt. ([Eff08])

oAW arbeitet bei der Generierung mit Templates. Für die Definition der Templates wird dabei die Template-Sprache *Xpand* verwendet. Listing 5-5 zeigt ein oAW-Template für ein EMF-Modell. Dieses Template generiert für jedes Modellelement `entity` eine Klasse. Für die Modellelemente `attribute` werden dabei private Felder erzeugt. ([Eff08])

```
1  «DEFINE Root FOR data::DataModel»
2    «EXPAND Entity FOREACH entity»
3  «ENDDEFINE»
4
5  «DEFINE Entity FOR data::Entity»
6     «FILE name + ".java"»
7        public class «name» {
8           «FOREACH attribute AS a»
9              private «a.type» «a.name»;
10          «ENDFOREACH»
11       }
12    «ENDFILE»
13 «ENDDEFINE»
```

Listing 5-5 Beispiel eines oAW-Templates zur Erzeugung von Java-Klassen, Quelle: ([Eff08])

Für die Validierung eines Modells stellt oAW die Sprache *Check* bereit. Mittels Check können Constraints formuliert werden. Listing 5-6 zeigt ein mit Check formulierten Constraint. Dieser

Constraint prüft, ob der Wert des Attributs `name` für jedes Modellelement `Attribute` länger als 1 Zeichen ist. Ist dies nicht der Fall, wird eine Fehlermeldung mit dem Text, der in Zeile 2 festgelegt ist, ausgegeben. ([Eff08])

```
1 context Attribute ERROR
2    "Names have to be more than one character long." :
3    name.length > 1;
```

Listing 5-6 Beispiel eines Constraints, formuliert mit Check, Quelle: ([Eff08])

5.3.3. AndroMDA

AndroMDA ist ein weiteres Open-Source-Generator-Framework. Es ist konzipiert für die Verwendung innerhalb des MDA-Entwicklungsprozesses. AndroMDA kann dabei Modelle verarbeiten, deren Metametamodell die MOF ist, beispielsweise UML-Modelle. Die Modelle können mittels AndroMDA validiert und transformiert werden. Die dafür erforderlichen Modell-zu-Modell-Transformationen können vom Entwickler in Java geschrieben werden. Für die Modell-zu-Code-Transformation stellt AndroMDA mehrere Cartridges zur Verfügung, beispielsweise für Hibernate oder EJB. Darüber hinaus kann AndroMDA durch eigene Cartridges erweitert werden. Abbildung 5-8 zeigt schematisch die Funktionsweise von AndroMDA. ([And08])

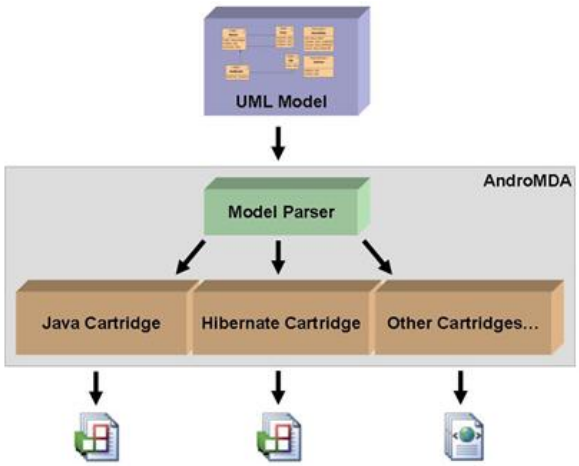


Abb. 5-8. AndroMDA, Quelle: ([Bha06])

Cartridges arbeiten bei AndroMDA Template-basiert. Standardmäßig wird dabei die Template-Engine *Velocity* verwendet. Velocity ist eine Open-Source-Template-Engine und ist implementiert in Java[12]. AndroMDA ermöglicht es jedoch auch, Velocity durch eine andere Template-Engine zu ersetzten. ([Pet06], Kap. 5.3)

[12]Mehr Informationen zu Velocity siehe http://velocity.apache.org/

Kapitel 6.

Anforderungsanalyse

Das im Rahmen der Diplomarbeit zu entwickelnde System soll das existierende System zur Umsetzung von virtuellen Ausstellungen am MPIWG ersetzen. Zum besseren Verständnis der an das zu entwickelnde System gestellten Anforderungen werden zunächst das existierende Altsystem und die verwendete Nomenklatur erläutert. Die Nomenklatur des Altsystems wird für das zu entwickelnde System übernommen. Im Anschluss an die Beschreibung des Altsystems werden dann die Probleme und Mängel, die bei der Verwendung des Altsystems festgestellt wurden, aufgeführt. Diese Mängel fließen zum Teil in die sich anschließende Definition der vor allem funktionalen Anforderungen mit ein. Diese Anforderungen wurden in Rücksprache mit dem Projektleiter Robert Casties festgelegt. Sie bilden die Grundlage für die Konzeption des zu entwickelnden Systems.

Neben den in diesem Kapitel definierten Anforderungen, wurden Wunschkriterien und Abgrenzungskriterien für das Zielsystem festgelegt. Diese befinden sich in Anhang C.2 und Anhang C.3. Des Weiteren wurde der Produkteinsatz spezifiziert. Die Spezifikationen befinden sich in Anhang C.4.

6.1. Beschreibung des Altsystems

Das existierende System zur Umsetzung von virtuellen Ausstellungen wurde mit *Zope* entwickelt. Zope ist ein in der Programmiersprache *Python* implementierter Open-Source-Applikationsserver. Er ist unter anderem für Unix-Betriebssysteme, Windows und Mac OS X verfügbar. Mittels Zope können komplexe Web-Anwendungen entwickelt werden. Hierfür kann der Funktionsumfang einer Zope-Installation durch zusätzliche Komponenten, sogenannte *Produkte*, erweitert werden. Eine Zope-Installation wird als *Zope-Instanz* bezeichnet. ([Hör03], Vorwort)

Die zur Zeit der Anfertigung dieser Diplomarbeit aktuelle Version von Zope ist Zope 3. Zope 3 wurde von Grund auf neu entwickelt und baut nicht auf Zope 2 auf. Die Softwarearchitektur von Zope 3 wurde in Bezug auf Zope 2 vollständig überarbeitet. Dadurch ist Zope 3 nicht

abwärtskompatibel mit Zope 2 ([Ric05], Kap. 12). Die am MPIWG für Zope entwickelten Produkte wurden jedoch für die Version 2 von Zope konzipiert. Das Altsystem wurde daher mit Zope 2 umgesetzt. Nachfolgend wird der Begriff "Zope" synonym für "Zope 2" verwendet.

Die Benutzerschnittstelle von Zope ist ein Web-Interface, das sogenannte *Zope Management Interface* (ZMI). Zope ist somit mittels eines Webbrowsers verwaltbar. Für die Kontrolle der Zugriffe auf das ZMI verfügt Zope über eine Benutzerverwaltung und ein umfassendes Sicherheitskonzept. Zope arbeitet dabei Objekt-basiert. Das bedeutet, dass die Elemente, die durch Zope erfasst und verwaltet werden können, durch Objekte dargestellt werden. Ein Verzeichnis in Zope wird beispielsweise durch ein Verzeichnis-Objekt repräsentiert. ([Hör03], Kap. 1 und 5)

Es gibt zwei unterschiedliche Möglichkeiten, ein Produkt für Zope zu erstellen: mit Hilfe von *ZClasses* oder mit Hilfe von *Python*. Eine ZClass ist ein Werkzeug, mit dem eine Klasse definiert werden kann. Von dieser Klasse können dann innerhalb von Zope Objekte angelegt und gespeichert werden. Ein Produkt, das mit ZClasses arbeitet, wird über das ZMI erstellt. Abbildung 6-1 zeigt das Zope Management Interface einer Zope-Instanz mit einer Auflistung aller installierten Produkte. ([Hör03], Kap. 14)

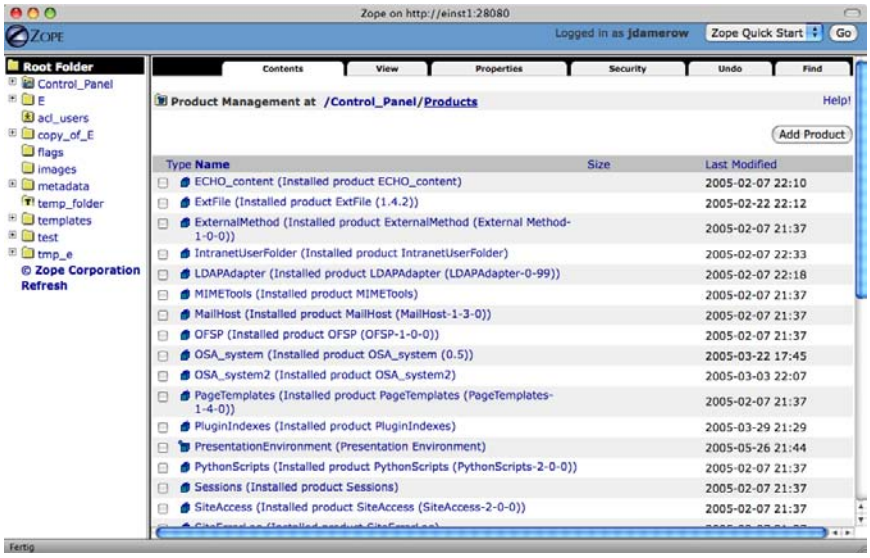


Abb. 6-1. Zope Management Interface

Ein Produkt, das mit Hilfe von Python erstellt wird, besteht aus Python-Klassen und -Skripten sowie Templates. Die Templates sind in den Template-Sprachen *DTML* (Document Template Markup Language) und *ZPT* (Zope Page Templates) verfasst. Die Templates und Python-Quelltexte eines Produktes müssen dabei, damit das Produkt verwendet werden kann, in einem speziellen Verzeichnis im Installationspfad einer Zope-Instanz vorhanden sein. ([Hör03], Kap. 14)

6.1.1. Nomenklatur

Eine virtuelle Ausstellung orientiert sich in Bezug auf Nomenklatur und Struktur am Aufbau einer realen Ausstellung. Dies ist darin begründet, dass das Konzept der virtuellen Ausstellung entwickelt wurde, um die "reale Einsteinausstellung" virtuell abzubilden. Inzwischen bilden virtuelle Ausstellungen neben realen Ausstellungen jedoch auch andere reale Räume ab oder haben keine real existierende Vorlage. Das Projekt "Pratolino: The History of Science in a Garden" beispielsweise ist die virtuelle Darstellung des in Italien gelegenen Gartens von Pratolino.[1] In Anhang B.1 finden sich einige Screenshots von virtuellen Ausstellungen. Im Folgenden werden die Begriffe und die Struktur einer virtuellen Ausstellung erläutert.

Virtual Exhibition

Eine *Virtual Exhibition* ist ein virtueller Rundgang. Eine Virtual Exhibition bildet einen realen oder abstrakten Raum ab. Im Folgenden werden die Begriffe "Virtual Exhibition" und "virtuelle Ausstellung" synonym zueinander verwendet.

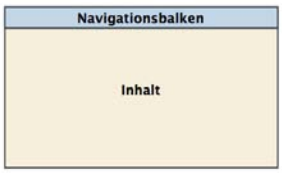


Abb. 6-2. Struktureller Aufbau einer Webseite

Eine Virtual Exhibition entspricht einer Website. Alle Webseiten dieser Website haben den gleichen strukturellen Aufbau. Am oberen Rand jeder Webseite befindet sich hierbei ein farblich unterlegter Balken. Dieser Balken enthält Informationen zur Webseite und dient teilweise der Navigation durch die virtuelle Ausstellung. Er wird im Folgenden als *Navigationsbalken* bezeichnet. Unter dem Navigationsbalken wird der Inhalt der Webseite dargestellt. Abbildung 6-2 zeigt schematisch den strukturellen Aufbau einer Webseite in einer virtuellen Ausstellung.

Scene

Eine *Scene* ist ein Abschnitt einer virtuellen Ausstellung. In der virtuellen Einsteinausstellung beispielsweise entspricht eine Scene einem Ausstellungsraum in der realen Einsteinausstellung. Der Einstiegspunkt in eine virtuelle Ausstellung wird durch die *Start-Scene* festgelegt.

Eine Scene wird dabei durch eine Webseite repräsentiert. Der Navigationsbalken der Webseite enthält beispielsweise den Titel einer Scene oder einen Verweis zur Start-Scene der virtuellen Ausstellung. Unter dem Navigationsbalken der Webseite wird ein Bild, beispielsweise von einem realen Ausstellungsraum, angezeigt. Dieses Bild wird im Folgenden als *Hintergrundbild* bezeichnet. Anhang B.2 zeigt beispielhaft zwei Scenes. Die Navigation durch die Scenes einer virtuellen Ausstellung erfolgt über Scene Links.

[1] Die virtuelle Ausstellung ist zu finden unter http://pratolino.mpiwg-berlin.mpg.de/.

Scene Link

Ein *Scene Link* ist ein Verweis von einer Scene zu einer anderen Scene. In einer virtuellen Ausstellung wird ein Scene Link durch einen anklickbaren Verweis in Form eines Textes oder eines Icons auf dem Hintergrundbild einer Scene dargestellt. Anhang B.3.2 zeigt eine Scene mit Scene Links.

Exhibition Module Link

Ein *Exhibition Module Link* ist ein Verweis von einer Scene zu einem *Exhibition Module*. Ein Exhibition Module ist dabei das virtuelle Abbild einer Medienstation in der Einsteinausstellung. In einer virtuellen Ausstellung wird ein Exhibition Module Link durch einen anklickbaren Verweis in Form eines Textes oder eines Icons repräsentiert. Dieser wird ebenso wie ein Scene Link auf dem Hintergrundbild einer Scene angezeigt. In Anhang B.3.2 befindet sich ein Screenshot einer Scene mit einem Exhibition Module Link.

Exhibition Module

Ein Exhibition Module stellt dem Besucher einer virtuellen Ausstellung Informationen zur Verfügung. Diese beziehen sich in der Regel auf das Thema der Scene, die auf das Exhibition Module verweist. Zu diesem Zweck umfasst ein Exhibition Module eine Reihe von Webseiten. Der Navigationsbalken dieser Webseiten enthält dabei Verweise zur Navigation durch ein Exhibition Module. Unter dem Navigationsbalken werden die Inhalte des Exhibition Modules angezeigt. Ein Exhibition Module ist aus den folgenden Bestandteilen aufgebaut:

- **Slides**
  Ein *Slide* ist die kleinste Einheit eines Exhibition Modules. Er entspricht einer Webseite.
- **Branching Points**
  Ein *Branching Point* ist eine spezielle Ausprägung eines Slides. Er stellt eine Verzweigung innerhalb der Navigation eines Exhibition Modules dar.
- **Sequences**
  Eine *Sequence* ist eine lineare Abfolge von Slides und Branching Points.
- **Meta-Sequences**
  Eine *Meta-Sequence* ist eine lineare Abfolge von Sequences.

Ein Exhibition Module ist ein Netzwerk von miteinander verbundenen Sequences. Dadurch entsteht bezüglich seiner Struktur ein gerichteter Graph, der Zyklen enthalten kann. Bei der Navigation durch ein Exhibition Module bewegt sich der Besucher einer virtuellen Ausstellung entlang der Kanten dieses Graphs. Der Graph wird daher im Folgenden als *Navigationsgraph*

bezeichnet. Zur Definition des Einstiegspunktes in ein Exhibition Module wird eine *Start-Sequence* definiert. Die Start-Sequence ist die Sequence, deren erstes Element bei Aufruf des Exhibition Modules angezeigt wird. In Abbildung 6-3 ist der Navigationsgraph eines Exhibition Modules dargestellt. Die einzelnen Elemente des Graphs werden im Folgenden detaillierter beschrieben.

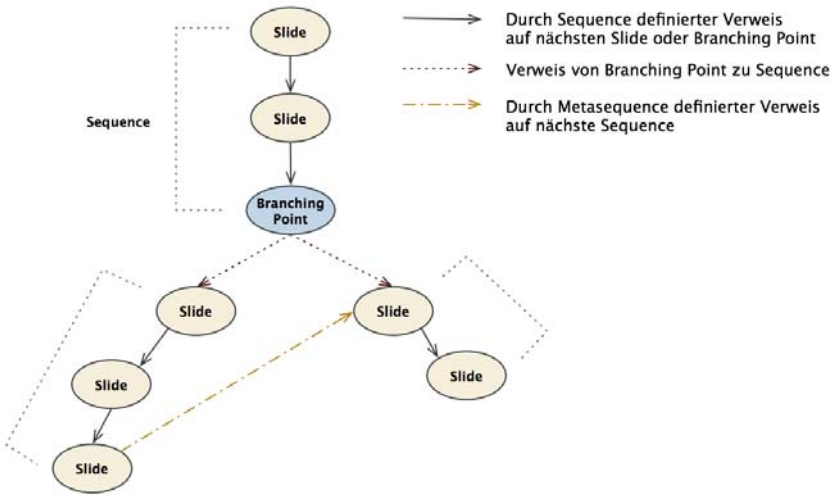


Abb. 6-3. Navigationsgraph eines Exhibition Modules

Sequence

Eine Sequence legt die Reihenfolge fest, in der Slides und Branching Points angezeigt werden. Ein Slide oder Branching Point gehört dabei genau einer Sequence an. Eine Sequence ist vergleichbar mit einer *PowerPoint*-Präsentation.

Meta-Sequence

Eine Meta-Sequence ist eine spezielle Ausprägung einer Sequence. Sie legt eine Reihenfolge fest, in der Sequences, beziehungsweise deren Inhalt, angezeigt werden. Auf eine Sequence kann von beliebig vielen Meta-Sequences verwiesen werden.

Slide

Die Slides eines Exhibition Modules enthalten die Informationen, die durch das Exhibition Module zur Verfügung gestellt werden. Ein Slide enthält zu diesem Zweck einen Text. Zusätzlich kann er ein Bild oder ein Video beinhalten. Der Inhalt des Slides wird, wie in Abbildung 6-4 dargestellt, unter dem Navigationsbalken angezeigt. Im Navigationsgraph eines Exhibition

Modules wird ein Slide durch einen Knoten repräsentiert. Ein Slide ist vergleichbar mit einer Folie in einer PowerPoint-Präsentation.

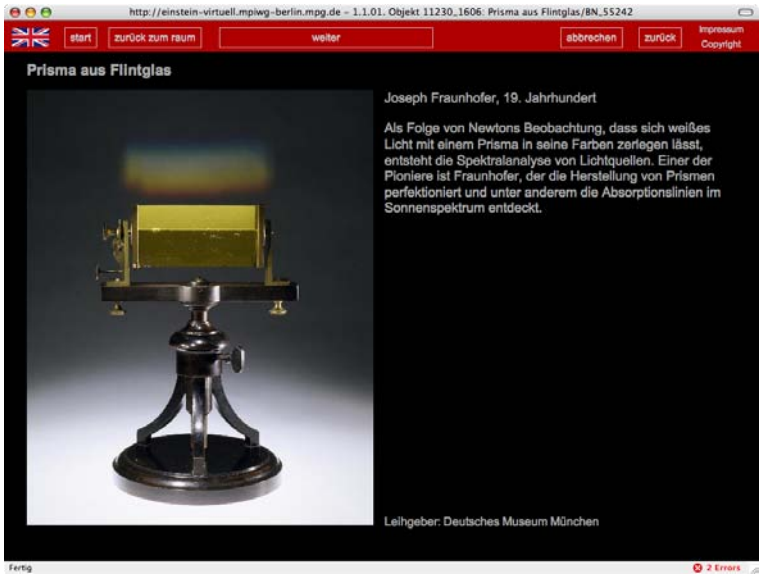


Abb. 6-4. Slide einer Medienstation

Branching Point

Ein Branching Point dient der Verzweigung innerhalb eines Navigationsgraphs. Er hat die Eigenschaften eines Slides, und darüber hinaus kann er Verweise zu Sequences oder Meta-Sequences enthalten. Ein Verweis zu einer Sequence oder Meta-Sequence wird dabei als *Branching Point Choice* bezeichnet. In Anhang B.3.1 befindet sich ein Screenshot eines Branching Points.

6.1.2. Implementierungsdetails

Das Altsystem ist mittels der Produkte *ECHO_content* und *PresentationEnvironment* umgesetzt. Das Produkt ECHO_content wurde von der IT-Abteilung des MPIWG entwickelt. Es wird für die Erstellung der Scenes verwendet. Dieses Produkt ist in Python implementiert. Ursprünglich wurde es für ein anderes Projekt entwickelt. Das Altsystem umfasst dadurch weit mehr Funktionen als für die Erstellung von Scenes benötigt werden. Die Benutzerschnittstelle des Produktes ECHO_content für die Verwaltung der Scenes ist das Zope Management Interface.

Das Produkt PresentationEnvironment wurde von Malcolm Hyman entwickelt. Es wird für die Erstellung von Exhibition Modules verwendet und wurde speziell für diesen Zweck entwi-

ckelt. Das Produkt wurde im Gegensatz zum Produkt ECHO_content mit ZClasses erstellt. Im Rahmen der Entwicklung wurde dabei eine speziell konzipierte Benutzerschnittstelle zur Verwaltung der Exhibition Modules implementiert. Diese kann mittels eines Webbrowsers bedient werden. Im Gegensatz zum ECHO_content-Produkt muss der Benutzer nicht das Zope Management Interface verwenden. In Anhang B.4 befinden sich Screenshots der Benutzerschnittstelle des Produktes PresentationEnvironment.

Zope ist ähnlich einem Filesystem organisiert. Der Inhalt einer Zope-Instanz ist somit baumartig strukturiert. Es gibt ein Root-Verzeichnis, in dem sich Unterverzeichnisse und Dateien befinden. Die Unterverzeichnisse enthalten wiederum Verzeichnisse und Dateien. Die Verzeichnisse und Dateien sind Objekte der Klassen, die in den Produkten definiert sind. ([Hör03], Kap. 4)

Bei dem Altsystem wird eine Scene durch ein Objekt repräsentiert, das die Eigenschaften eines Verzeichnisses besitzt. Das Objekt kann andere Objekte, beispielsweise andere Scenes, enthalten. Die Baumstruktur von Zope wird dabei zur Definition der Scene Links genutzt. Dieses Konzept ist in Abbildung 6-5 dargestellt. Jede Scene einer virtuellen Ausstellung erhält automatisch einen Verweis zu der vorhergehenden Scene. Die vorhergehende Scene ist durch das Elternobjekt des Scene-Objekts innerhalb der Baumstruktur von Zope definiert. Die nachfolgenden Scenes einer Scene sind durch die Kinderobjekte des Scene-Objekts festgelegt. Scene Links zwischen Scenes aus unterschiedlichen Zweigen der Baumstruktur werden mittels Objekten realisiert, die nach dem Vorbild von HTML-Links konzipiert sind.

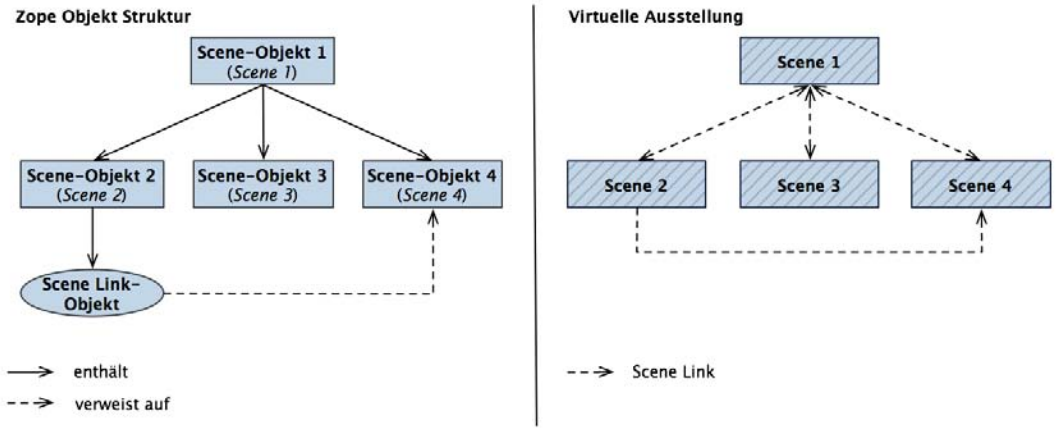


Abb. 6-5. Realisierung der Scene Links

Eine der mächtigsten Mechanismen von Zope ist die sogenannte *Akquisition*. Das bedeutet, dass ein Objekt alle Methoden und Eigenschaften der in der Baumstruktur vorhergehenden Objekte erbt. Abbildung 6-6 erläutert dieses Konzept. ([Hör03], Kap. 4) Der Mechanismus der Akquisition wird für das Layout einer virtuellen Ausstellung angewandt. Dieses wird durch Templates festgelegt. Die Templates sind dabei in der Template-Sprache ZPT verfasst. Sie

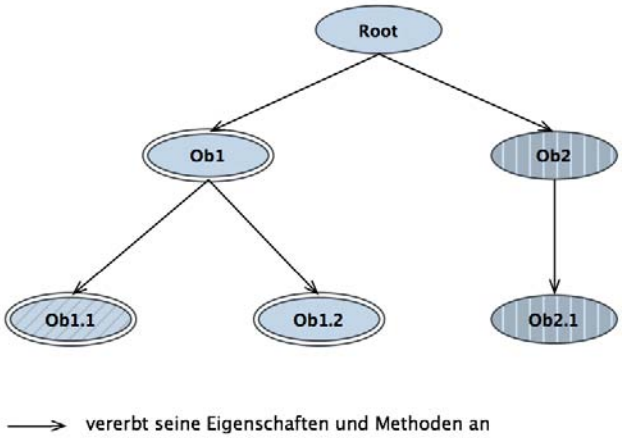


Abb. 6-6. Akquisition

befinden sich auf der obersten Ebene einer virtuellen Ausstellung innerhalb der Baumstruktur von Zope. Durch die Akquisition können sie auf alle darunterliegenden Objekte angewandt werden. In Anhang B.4.4 befindet sich ein ZPT-Template zur Anzeige einer Scene.

Zur Verwaltung der in einer virtuellen Ausstellung verwendeten Bilder wird eine *ImageCollection* verwendet. Eine ImageCollection ist eine Web-Anwendung, die am MPIWG entwickelt wurde. Sie dient der Erfassung und Verwaltung von Bildern und deren Metadaten. Eine ImageCollection bietet dafür grundlegende Funktionen zur Bearbeitung von Bildern, wie beispielsweise Funktionen zur Kontrastveränderung oder zum Drehen der Bilder. Darüber hinaus verfügt eine ImageCollection über eine mächtige Funktionalität zur Skalierung von Bildern. Jede Bildbearbeitung, die mittels einer ImageCollection durchgeführt wird, verändert das Original des Bildes dabei jedoch nicht. Die Bedienung einer ImageCollection erfolgt mittels eines Webbrowsers. In Anhang B.4.5 befindet sich ein Screenshot einer ImageCollection.

6.2. Mängel des Altsystems

Die bei der Benutzung des Altsystems aufgetretenen Probleme und Mängel sollen durch das im Rahmen dieser Diplomarbeit zu entwickelnde System behoben werden. Diese nachfolgend beschriebenen Probleme und Mängel betreffen hauptsächlich den Software-ergonomischen Aspekt und die technischen Eigenschaften des Altsystems.

- Für die Erstellung und Verwaltung von Scenes sind Kenntnisse in der Verwendung von Zope notwendig. Die Benutzerschnittstelle ist dabei das ZMI. Die Verwaltung von Scenes setzt somit Kenntnisse im Umgang mit dieser Schnittstelle voraus.
- Es gibt keine speziell für die Verwaltung von Scenes konzipierte Benutzer-Schnittstelle. Das Produkt ECHO_content stellt daher mehr Funktionen als für die Verwaltung von

Scenes benötigt zur Verfügung. Die Bedienung des Altsystems wird somit durch das Überangebot von Funktionalität erschwert.

- Für die Anpassung des Layouts einer virtuellen Ausstellung sind Kenntnisse in der Template-Sprache ZPT notwendig.
- Bei einem Wechsel der Version von Zope, beispielsweise von Zope 2.8 auf Zope 2.9, entstehen Probleme bei dem Altsystem. Dies ist darin begründet, dass das Produkt PresentationEnvironment mittels ZClasses erstellt wurde. Häufig gibt es in den Versionen geringe Unterschiede, die jedoch bei der Verwendung von ZClasses zu erheblichen Problemen führen können. Bei der Erstellung von Produkten mittels Python hingegen ist ein Wechsel der Version weniger problematisch.
- Eine Übersicht über die gesamte virtuelle Ausstellung ist nicht möglich. Für die Verwaltung der Exhibition Modules gibt es zwar Übersichten, beispielsweise zur Anzeige aller Exhibition Modules oder aller Sequences in einem Exhibition Module. Es gibt jedoch keine Übersicht über alle Scenes, Scene Links und Exhibition Module Links.
- Ein Objekt in Zope wird über den Pfad vom Root-Verzeichnis zu dem Objekt identifiziert. Dieser Pfad wird von Scene Links und Exhibition Module Links als Verweis auf eine Scene oder ein Exhibition Module genutzt. Er ändert sich jedoch, wenn die Lage eines Objekts innerhalb der Baumstruktur von Zope verändert wird. Bei der Verschiebung eines Scene-Objekts oder Exhibition Module-Objekts innerhalb von Zope werden die auf diese Scene oder das Exhibition Module verweisenden Scene Links beziehungsweise Exhibition Module Links nicht aktualisiert. Die Verwendung der Scene Links oder Exhibition Module Links führt daher zu Fehlern.
- Ein Slide oder Branching Point kann genau ein Bild enthalten. Bei der Umsetzung von virtuellen Ausstellungen hat sich jedoch herausgestellt, dass es häufig erforderlich ist, mehr als ein Bild in einem Slide oder Branching Point anzuzeigen.

6.3. Anforderungen

Das im Rahmen dieser Diplomarbeit zu konzipierende System wird im Folgenden als *die Ausstellungssoftware* bezeichnet. An den für die Ausstellungssoftware entwickelten Prototyp werden dabei mehrere Anforderungen gestellt. Diese werden im Folgenden zusammenfassend aufgeführt. Eine detaillierte Beschreibung der Anforderungen befindet sich in Anhang C.1.

6.3.1. Virtuelle Ausstellung

Der Prototyp soll das Erstellen, Ändern und Löschen einer virtuellen Ausstellung ermöglichen. Dies beinhaltet die in Abbildung 6-7 dargestellten Use-Cases.

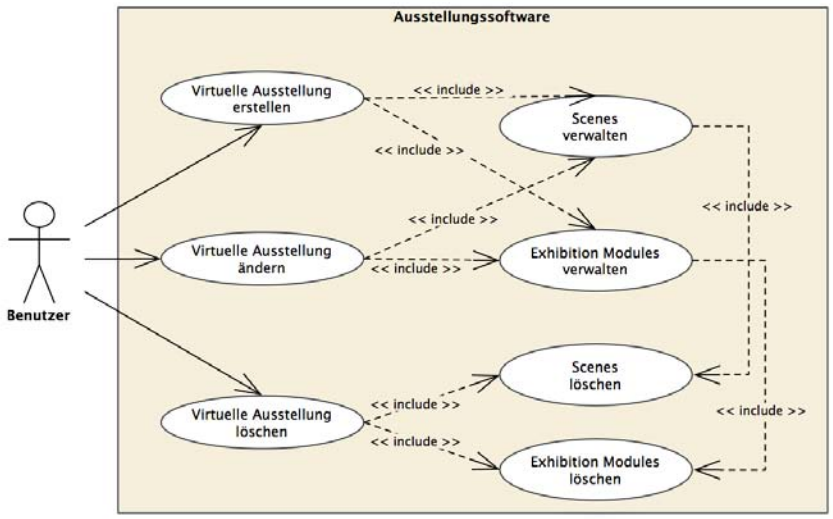


Abb. 6-7. Use-Cases Virtuelle Ausstellung

Die Anforderungen, die an den Prototyp diesbezüglich gestellt werden, sind nachfolgend aufgeführt.

- Für das Erstellen, Ändern und Löschen einer virtuellen Ausstellung soll der Prototyp die Verwaltung von Scenes und Exhibition Modules ermöglichen. Einer virtuellen Ausstellung sollen dafür Scenes oder Exhibition Modules hinzugefügt, geändert und entfernt werden können.

- Wird eine virtuelle Ausstellung gelöscht, sollen dabei alle darin enthaltenen Scenes und Exhibition Modules ebenfalls gelöscht werden.

- Eine mit der Ausstellungssoftware erstellte virtuelle Ausstellung soll mit Hilfe von HTML-Seiten und JavaScript im Internet präsentiert werden können. Für die Anwendung der Ausstellungssoftware sollen jedoch keine Kenntnisse in der Programmierung mit HTML und JavaScript erforderlich sein. Der Prototyp soll daher die Webseiten für eine virtuelle Ausstellung automatisch erzeugen.

- Die Ausstellungssoftware soll eine virtuelle Ausstellung möglichst so darstellen, wie die Ausstellung durch die Webseiten im Internet präsentiert wird. Das bedeutet unter anderem, dass die konkrete Syntax der DSL möglichst intuitiv anwendbar ist.

- Die Anwender der Ausstellungssoftware haben keine Ausbildung in Webdesign und auch in der Regel wenig gestalterisches Hintergrundwissen. Es soll daher ein Standard-Layout für die Webseiten einer virtuellen Ausstellung geben, das nur bei Bedarf angepasst wird. Ein Benutzer kann somit eine virtuelle Ausstellung erstellen, ohne layoutbezogene Konfigurationen vorzunehmen.

- Das Design der Webseiten einer virtuellen Ausstellung soll zentral veränderbar sein. Es kann beispielsweise durch CSS-Dateien festgelegt werden. Für das Ändern des Designs können spezifische Kenntnisse, beispielsweise von CSS, von dem Anwender gefordert werden.

- Das Layout von Slides und Branching Points soll über Vorlagen geregelt werden. Die Vorlagen definieren dabei die Anordnung von Texten und Bildern. Der Benutzer kann sich somit ausschließlich auf die inhaltlichen Aspekte einer virtuellen Ausstellung konzentrieren.

6.3.2. Verwaltung von Scenes

Die Verwaltung der Scenes soll das Erstellen, Ändern und Löschen von Scenes und das Erstellen, Ändern und Löschen von Scene Links umfassen. Die entsprechenden Use-Cases sind in Abbildung 6-8 dargestellt.

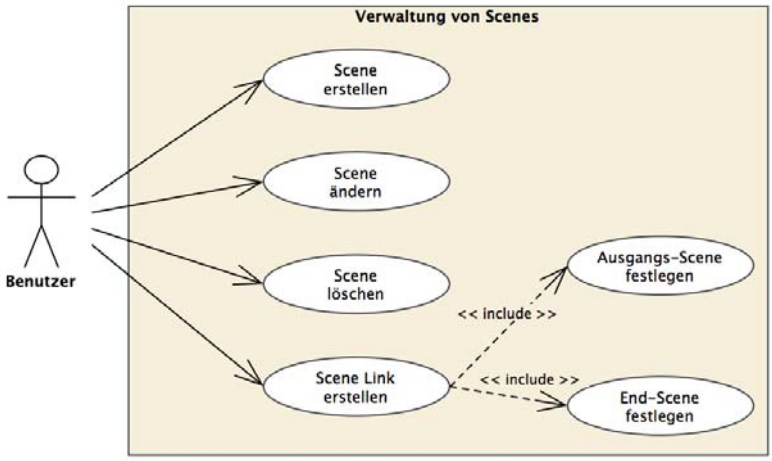


Abb. 6-8. Use-Cases Scenes

Die an den Prototyp gestellten Anforderungen für die Verwaltung der Scenes sind im Folgenden aufgeführt.

- Bei der Erstellung einer Scene soll der Scene ein Bild zugewiesen werden können. Das Bild dient dabei in der Webseite der Scene als Hintergrundbild, auf dem Scene Links und Exhibition Module Links positioniert sind. Das Hintergrundbild soll jederzeit geändert und gelöscht werden können.

- Von einer Scene soll auf andere Scenes mittels Scene Links verwiesen werden können. Ein Scene Link soll dabei unidirektional sein. Für einen Scene Link sollen hierfür ei-

ne Ausgangs-Scene und eine End-Scene festgelegt werden können. Abbildung 6-9 zeigt schematisch einen Scene Link.

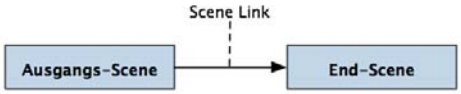


Abb. 6-9. Scene Link

- Das Löschen einer Scene soll dazu führen, dass alle Scene Links, bei denen die Scene Ausgangs- oder End-Scene ist, ebenfalls gelöscht werden.

6.3.3. Verwaltung von Exhibition Modules

Die Verwaltung der Exhibition Modules soll das Erstellen, Ändern und Löschen von Exhibition Modules und das Erstellen, Ändern und Löschen von Exhibition Module Links beinhalten. Die dafür relevanten Use-Cases sind in Abbildung 6-10 dargestellt.

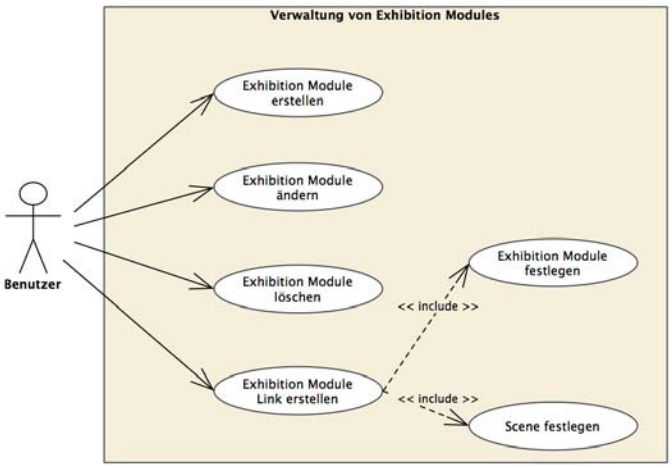


Abb. 6-10. Use-Cases Exhibition Module

Die bezüglich der Verwaltung von Exhibition Modules an den Prototyp gestellten Anforderungen werden nachfolgend erläutert.

- Einem Exhibition Module sollen Slides, Branching Points und Sequences hinzugefügt werden können. Diese Slides, Branching Points und Sequences sollen darüber hinaus geändert und gelöscht werden können.

- Auf ein Exhibition Module soll von einer Scene mittels eines Exhibition Module Links verwiesen werden können. Der Verweis soll hierbei, wie in Abbildung 6-11 dargestellt, bidirektional sein.

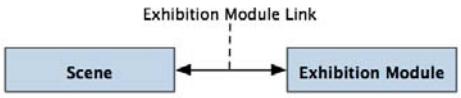


Abb. 6-11. Exhibition Module Link

- Das Löschen eines Exhibition Modules soll dazu führen, dass alle Slides, Branching Points und Sequences des Exhibition Modules ebenfalls gelöscht werden. Darüber hinaus sollen dabei die Exhibition Module Links, mittels derer von den Scenes auf das Exhibition Module verwiesen wird, ebenfalls gelöscht werden.
- Für ein Exhibition Module soll eine Start-Sequence festgelegt und jederzeit geändert werden können.
- Slides und Branching Points sollen einen Titel, einen Untertitel und eine Fußzeile haben können. Darüber hinaus sollen ihnen im Gegensatz zum Altsystem beliebig viele Texte und Bilder zugewiesen werden können.
- Ein Branching Point soll darüber hinaus Branching Point Choices beinhalten. Die Branching Point Choices sollen beim Löschen des Branching Points gleichermaßen gelöscht werden.
- Sequences sollen durch Verweise auf Slides und Branching Points die Reihenfolge festlegen, in der die Slides und Branching Points angezeigt werden. Des Weiteren sollen die Verweise von Sequences auf einen Slide oder Branching Point gelöscht werden, wenn der entsprechende Slide oder Branching Point gelöscht wird.

6.3.4. Software-Ergonomie

Neben den zuvor aufgeführten, zum großen Teil funktionalen Anforderungen werden einige Anforderungen an den Prototyp gestellt, die die Software-Ergonomie betreffen. Diese sind nachfolgend aufgeführt.

- Am MPIWG arbeiten neben deutschsprachigen Wissenschaftlern Gastwissenschaftler aus nicht deutschsprachigen Ländern. Des Weiteren finden häufig Kooperationen mit nicht deutschsprachigen Institutionen statt. Die Arbeitssprache am MPIWG ist daher neben Deutsch auch Englisch. Die Ausstellungssoftware soll daher über eine englische Benutzeroberfläche verfügen.

- Die Anwender der Ausstellungssoftware haben wenig bis gar keine software-technischen Kenntnisse. Die Ausstellungssoftware soll daher ohne software-technisches Hintergrundwissen bedienbar und erlernbar sein.

6.4. Abgrenzung

Bei der Konzeption des Prototyps steht die Modellierung und anschließende Generierung im Vordergrund. Das bedeutet, dass die konkrete Umsetzung der erzeugten Webseiten für diese Diplomarbeit nicht von Bedeutung ist. Ein Beispiel für die Website einer virtuellen Ausstellung befindet sich auf der der Arbeit beiliegenden CD-ROM. Des Weiteren dient der Prototyp in erster Linie der Untersuchung, ob der gewählte Ansatz der modellgetriebenen Softwareentwicklung für die Umsetzung einer Ausstellungssoftware, eines *Tools*, anwendbar ist. Über die zuvor aufgeführten Anforderungen hinausgehende Funktionen, wie beispielsweise die Verwaltung der generierten Webseiten, sind daher für den Prototyp nicht vorgesehen.

Kapitel 7.

Konzeption und Umsetzung des Systems

Im Folgenden werden die Konzeption und Umsetzung des im Rahmen dieser Diplomarbeit entwickelten Prototyps zur Erstellung von virtuellen Ausstellungen beschrieben. Die Projektbezeichnung für den Prototyp ist dabei *JAT* (*JAT Ausstellungs Tool*). JAT ist ein rekursives Akronym. Ein rekursives Akronym ist ein Akronym, dessen Erklärung sich selbst enthält. Rekursive Akronyme treten meist bei Open-Source-Projekten auf. Der Name des Prototyps wurde in Anlehnung an die Technologie gewählt, mit der das Altsystem umgesetzt wurde. Diese war Zope, was für Z(ope) Object Publishing Environment steht.

Der Prototyp soll den im vorherigen Kapitel definierten Anforderungen entsprechen und wenn zeitlich möglich einige der in Anhang C.2 festgelegten Wunschkriterien erfüllen. Im Hinblick darauf werden zunächst die Techniken für die Umsetzung der Ausstellungssoftware ausgewählt. Im Anschluss daran werden die ausgewählten Techniken kritisch betracht. In der folgenden Beschreibung der Konzeption und Umsetzung des Prototyps wird zunächst die Konstruktion des Domänenmodells und des Metamodells erläutert. Anschließend folgt eine Darstellung der Umsetzung des Modellierungstools und der Realisierung des Codegenerators. Schließlich wird auf einige Implementierungsdetails näher eingegangen. Zuletzt werden die eingesetzten Teststrategien kurz erläutert.

7.1. Auswahl der Techniken

Für die Wahl der Techniken zur Umsetzung des Prototyps gibt es drei grundlegende Bedingungen.

- In Absprache mit dem Projektleiter Robert Casties wurde als Programmiersprache Java ausgewählt. Dies hat den Grund, dass die verwendeten Programmiersprachen am MPIWG hauptsächlich Java und Python sind.

- Die MPG ist ein engagierter Vertreter der "Open Access"-Politik[1]. Daher besteht die Bedingung, dass die eingesetzten Techniken frei und wenn möglich open-source verfügbar sind.

- Die zu entwickelnde Software muss auf dem Betriebsystem OS X von Apple lauffähig sein. Dies ist darin begründet, dass am MPIWG fast ausschließlich mit diesem Betriebssystem gearbeitet wird.

7.1.1. Modellierung

Bei der Wahl des Modellierungswerkzeugs werden auf Grund der "Open Access"-Politik des MPIWG die kommerziellen Werkzeuge ArcStyler und Enterprise Architect nicht in den Auswahlprozess miteinbezogen. Das *Generic Modeling Environment* wird ebenfalls ausgeschlossen. Es ist zwar open-source verfügbar, es wurde jedoch für das Betriebssystem Windows entwickelt. Der Vergleich von GMF und Xtext ergibt, dass beide Frameworks in Java implementiert und somit auf dem Betriebssystem OS X lauffähig sind. Bei der Verwendung von GMF kann dabei eine graphische DSL definiert werden. Xtext arbeitet dagegen mit textuellen DSLs. Eine graphische DSL ist für einen Benutzer ohne technisches Hintergrundwissen, wie beispielsweise das Beherrschen einer Programmiersprache, in der Regel einfacher zu verwenden als eine textuelle DSL. Die Wahl des Modellierungswerkzeugs fällt daher auf GMF.

Die Verwendung von GMF zur Modellierung legt EMF als das Metamodellierungswerkzeug und Ecore als Metametamodell fest. Ecore ermöglicht die Erstellung von Metamodellen, die spezifisch für eine Domäne sind. Die Domäne einer virtuellen Ausstellung ist keine technische Domäne wie beispielsweise die Architektur einer Software. Es bietet sich daher an, nicht die UML zu verwenden, sondern dem Ansatz von Ecore zu folgen. EMF ist somit für die Definition des Metamodells gut geeignet.

Die Verwendung von GMF setzt bei der Entwicklung des Prototyps den Einsatz der IDE Eclipse voraus. Eclipse ist aus mehreren Ebenen aufgebaut. Die Kern-Komponente ist hierbei die *Eclipse Runtime Platform*. Sie stellt grundlegende Funktionen, wie beispielsweise die Plugin-Verwaltung oder die Verwaltung von Dateien und Verzeichnissen, bereit. Darauf aufbauend fügt, wie in Abbildung 7-1 dargestellt, die *Eclipse Integrated Development Environment* wesentliche Funktionen, die eine IDE benötigt, hinzu. Zu diesen Funktionen gehören beispielsweise Suchfunktionen oder Debugging-Funktionen. Die *Java Development Tools* (JDT) erweitern Eclipse darüber hinaus um notwendige Funktionen zur Entwicklung von Java. Weitere Funktionalität wird über Plugins hinzugefügt. Diese Anreicherung der Funktionalität durch Plugins ist das Kernkonzept von Eclipse. Abbildung 7-1 zeigt die Architektur von Eclipse. ([Car05], Kap. 1.1)

[1] Mehr zur "Open Access"-Politik der MPG siehe http://oa.mpg.de/openaccess-berlin/berlindeclaration.html

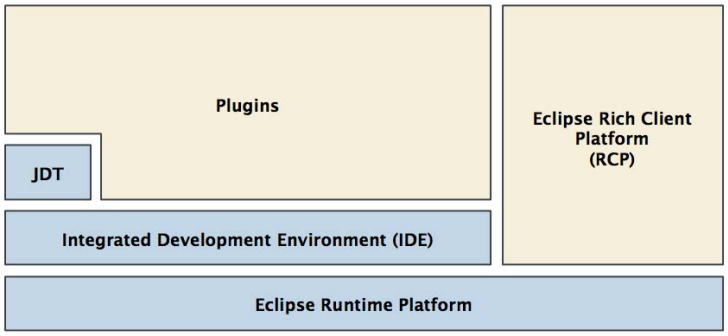


Abb. 7-1. Architektur von Eclipse, Quelle: angelehnt an ([Car05], Kap. 1.1)

Neben der Verwendung und Erweiterung der Entwicklungsumgebung können auf Basis der Eclipse Runtime Platform Rich-Client-Anwendungen entwickelt werden. Eine solche Rich-Client-Anwendung wird als *Eclipse Rich-Client-Platform* (Eclipse RCP) bezeichnet. Eclipse RCPs folgen ebenso wie die Entwicklungsumgebung dem Plugin-Konzept. Zusätzliche Funktionalität kann dem Eclipse RCP daher über Plugins hinzugefügt werden. ([Car05], Kap. 1.1)

Ein mit GMF erstelltes Modellierungstool kann als Plugin für Eclipse oder als Eclipse RCP erzeugt werden. Die Anwender der Ausstellungssoftware sind in der Regel Geschichtswissenschaftler, die keine Kenntnisse in der Verwendung einer IDE besitzen. Die Entwicklung des GMF-Modellierungstools als Plugin für Eclipse würde jedoch Kenntnisse im Umgang mit Eclipse voraussetzen. Darüber hinaus ständen bei der Nutzung von Eclipse ähnlich wie beim Altsystem mehr Funktionen zur Verfügung, als für die Erstellung einer virtuellen Ausstellung notwendig sind. Es wurde daher entschieden, das GMF-Modellierungstool als Eclipse RCP zu entwickeln. Die Ausstellungssoftware kann somit an die benötigte Funktionalität angepasst werden.

7.1.2. Codegenerator

In den Auswahlprozess des Codegenerators werden die Generatoren JET, openArchitectureWare und AndroMDA aufgenommen. Die drei Generatoren sind Open-Source-Frameworks und auf OS X lauffähig. AndroMDA ist für die Verwendung innerhalb eines MDA-Entwicklungsprozesses konzipiert. Es existieren Cartridges für unterschiedliche Technologien, wie beispielsweise für die Generierung von Java-Klassen. Für die Erzeugung von HTML-Seiten bietet AndroMDA jedoch keine Unterstützung. Des Weiteren kann bei der Betrachtung der Übersicht über die zuletzt veröffentlichten Versionen von AndroMDA festgestellt werden, dass die letzte stabile Version 2006 herausgegeben wurde.[2] Dies führt zu der Annahme, dass die Entwicklung von AndroMDA stagniert oder eventuell ganz eingestellt wurde. Die Verwendung

[2] Siehe hierzu http://galaxy.andromda.org/docs/changes-report.html.

von AndroMDA könnte somit in Entwicklungsphasen, die auf den Prototyp folgen, dazu führen, dass auf einen anderen Codegenerator umgestellt werden muss. AndroMDA wird für die Umsetzung des Prototyps daher nicht verwendet.

Im Gegensatz zu AndroMDA sind oAW und JET nicht auf MDA spezialisiert. oAW umfasst dabei neben der M2C-Transformation und M2M-Transformation auch die Möglichkeit der Erstellung von textuellen DSLs und einen Metamodell-Generator, der aus UML-Modellen Ecore-Modelle erstellt. Des Weiteren ermöglicht oAW das Validieren von Modellen und die Verwendung von unterschiedlichen Modellierungswerkzeugen. Die umfassende Funktionalität von oAW führt zu der Entscheidung, oAW nicht für die Umsetzung des Prototyps zu verwenden. Die Generierung der Webseiten für eine virtuelle Ausstellung würde nur einen kleinen Teil der Funktionalität von oAW nutzen.

JET verfügt im Vergleich zu oAW über weit weniger Funktionalität. Dieser Codegenerator ist auf die Generierung von Java-Quelltext spezialisiert. Bei der Generierung von Java-Code kann beispielsweise JMerge bei der Zusammenführung von generiertem und handgeschriebenem Quelltext verwendet werden. Bei der Generierung von HTML hingegen müssen Protected Regions im Template explizit ausgezeichnet werden. Dieses Vorgehen ist fehleranfälliger und weniger komfortabel als die Verwendung von JMerge. Des Weiteren haben erste Tests ergeben, dass JET nicht ermöglicht, mehr Informationen in den Transformationsprozess einfließen zu lassen als im Modell verfügbar sind. Bei der Erzeugung der HTML-Seiten für eine virtuelle Ausstellung werden jedoch neben den Informationen aus dem Modell Informationen zur Darstellung von Modellelementen benötigt. Es werden beispielsweise Daten zur Positionierung von Scene Links und Exhibition Module Links benötigt. Diese Daten sind nicht im Modell gespeichert. Bei der Verwendung von JET müssten somit zusätzliche Daten zunächst im Modell verfügbar gemacht werden. JET wird daher zur Umsetzung des Prototyps ebenfalls ausgeschlossen.

7.1.3. Überblick der ausgewählten Techniken

Der Vergleich der Umsetzungsmöglichkeiten hat ergeben, dass der Prototyp mittels GMF entwickelt wird. Im Folgenden wird der Prototyp auch als *das Modellierungstool* bezeichnet. Im Gegensatz dazu wird der Begriff "GMF-Modellierungstool" verwendet, wenn eine Aussage für jedes mit GMF erstellte Modellierungstool gültig ist. Des Weiteren wird für die Entwicklung Ecore als Metametamodell verwendet. Der Vergleich der Codegeneratoren hat ergeben, dass weder AndroMDA, noch oAW oder JET für die Umsetzung des Prototyps geeignet sind. Der Codegenerator wird daher speziell für den Prototyp konzipiert und implementiert. Abbildung 7-2 zeigt das Zusammenwirken der verwendeten Technologien.

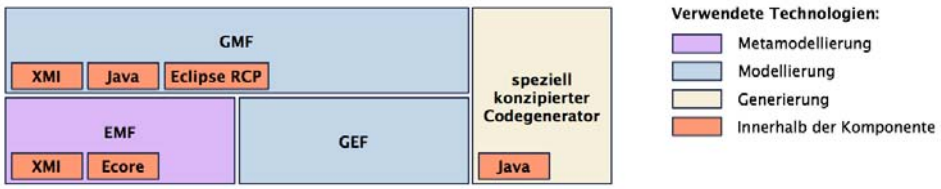


Abb. 7-2. Verwendete Technologien

7.1.4. Kritische Betrachtung der Techniken

Das Altsystem wird ausschließlich über das Internet mittels eines Webbrowsers verwendet. Dies hat den Vorteil, dass mehrere Wissenschaftler gleichzeitig und von unterschiedlichen Orten aus an einer virtuellen Ausstellung arbeiten können. Insbesondere bei der Erstellung der Exhibition Modules der virtuellen Einsteinausstellung wurde diese Möglichkeit genutzt. Ein GMF-Modellierungstool ist im Gegensatz zum Altsystem eine lokale Anwendung. Die Zusammenarbeit von unterschiedlichen Personen ist dadurch zunächst nicht möglich. Bei der virtuellen Einsteinausstellung nachfolgenden Projekten, bei denen eine virtuelle Ausstellung erstellt wurde, arbeitete jedoch in der Regel nur eine Person an einer virtuellen Ausstellung. Eine lokale Anwendung ist daher in den meisten Fällen ausreichend.

Die Zusammenarbeit von unterschiedlichen Personen bei der Erstellung einer virtuellen Ausstellung könnte durch die Verwendung eines Versionskontrollsystems zur Speicherung der von dem Modellierungstool benötigten Dateien ermöglichen werden. Darüber hinaus könnte ein Redaktionsprozess durch den Einsatz eines Test- und eines Produktionssystems möglich gemacht werden. Änderungen an einer virtuellen Ausstellung würden dabei zunächst in das Testsystem eingepflegt werden. Erst durch die Freigabe des Projektleiters würden die Änderungen in das Produktionssystem übernommen werden.

7.2. Entwicklungssystem

Der Prototyp wurde unter Verwendung des Betriebssystems OS X von Apple und der IDE Eclipse entwickelt. Die verwendete Version des *Java SE Development Kit* (JDK) ist das JDK 5.[3] Für die Versionierung des Prototyps wurde *Subversion* (SVN) genutzt. Subversion ist ein Open-Source-Versionskontrollsystem.[4] Für das Projektmanagement wurde das Web-basierte Projektmanagement-Tool *Trac* verwendet.[5]

[3] Weitere Informationen zu JDK 5 siehe http://java.sun.com/javase/downloads/index_jdk5.jsp.
[4] Weitere Informationen zu Subversion siehe http://subversion.tigris.org/.
[5] Weitere Informationen zu Trac siehe http://trac.edgewall.org/.

7.3. Produktivsystem

Für den Einsatz des zu entwickelnden Systems wird das Betriebssystem OS X von Apple vorausgesetzt. Des Weiteren wird die *Java Runtime Environment* (JRE) in der Version 5.0 oder höher benötigt.[6]

7.4. Vorgehensmodell

Für die Entwicklung des Prototyps wurde das Vorgehensmodell *Crystal Clear* gewählt. Crystal Clear gehört zu einer Gruppe von Vorgehensmodellen, die *Crystal* genannt wird. Crystal wurde von Alistair Cockburn entwickelt.

Die Methoden der Crystal-Vorgehensmodelle sind für die unterschiedlichen Voraussetzungen, die für ein Projekt gelten können, konzipiert. Crystal Clear wurde dabei für den Einsatz bei Projekten mit einem Entwicklerteam, das bis zu sechs Personen umfasst, entwickelt. Crystal Clear legt die folgenden Prinzipien für die Softwareentwicklung fest. ([Coc06], Kap. 6)

- Die Entwickler befinden sich in einem Raum oder in angrenzenden Räumen. Dies dient der Förderung der Kommunikation zwischen den Entwicklern.
- Der Auftraggeber erhält regelmäßig, etwa alle zwei bis drei Monate, eine neue Version der zu entwickelnden Software.
- Der Auftraggeber bzw. der spätere Anwender der Software wird in den Entwicklungsprozess miteinbezogen.
- Jede ausgelieferte Version wird von dem späteren Anwender getestet.
- Zu Beginn und in der Mitte des Entwicklungszeitraumes einer Version werden Besprechungen zur Verbesserung der Software und der Vorgehensweise bei der Entwicklung abgehalten.
- Ein Versionskontrollsystem wird verwendet.
- Neben den ausgelieferten Versionen, dem Code, Software-Tests und Use-Case-Diagrammen werden nur wenige Dokumente während der Entwicklung erstellt.

Bei der Entwicklung des Prototyps bestand das Entwicklerteam aus einer Person. Ansprechpartner wie Fachexperten oder der Projektleiter befanden sich im gleichen Raum wie die Entwicklerin oder in sehr nahegelegenen Büroräumen. Des Weiteren wurden zu Beginn der Entwicklungsphase des Prototyps Besprechungen zur Anforderungsdefinition und zur Klärung von Fachfragen geführt. Ebenso wurden während der Entwicklung Sitzungen zur Beurteilung des Prototyps organisiert. Die räumliche Nähe zu den Fachexperten führte regelmäßig zu spontanen Besprechungen hinsichtlich der Konzeption und Bedienbarkeit des Prototyps.

[6] Weitere Informationen zu JRE 5 siehe http://java.sun.com/javase/downloads/index_jdk5.jsp.

7.5. Metamodellierung

Die Definition des Metamodells ist ein wesentlicher Bestandteil der modellgetriebenen Softwareentwicklung. Die Struktur der Domäne muss dabei möglichst genau abgebildet werden. Änderungen, die nach der Metamodellierungsphase am Metamodell vorgenommen werden, führen insbesondere bei der Verwendung von GMF oftmals zu Schwierigkeiten. Der Grund dafür liegt darin, dass die Entwicklung eines GMF-Modellierungstools auf dem mit EMF erstellten Metamodell aufbaut.

7.5.1. Definition des Domänenmodells

Das Domänenmodell des Prototyps wurde auf dem Domänenmodell des Altsystems basierend entwickelt. Das Domänenmodell des Altsystems wurde hierfür unter Einbeziehung der Erfahrungen aus der Verwendung des Altsystems überarbeitet. Abbildung 7-3 zeigt das Domänenmodell des Prototyps. Dieses wird im Folgenden erläutert. In Anhang D.1 befindet sich eine detailliertere Version des Domänenmodells.

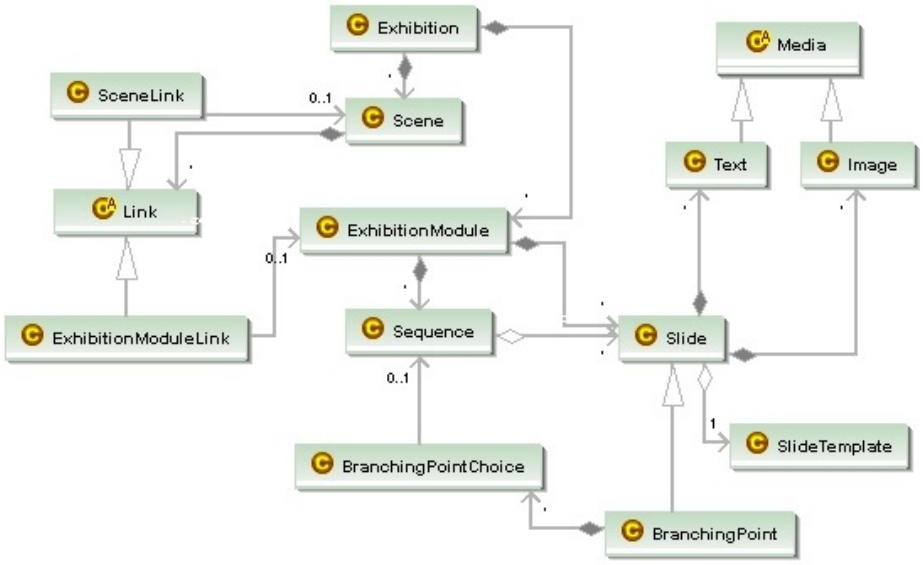


Abb. 7-3. Domänenmodell des Prototyps

Das Domänenmodell des Prototyps besteht zunächst aus den drei zentralen Elementen einer virtuellen Ausstellung:

- die Virtual Exhibition, abgebildet durch die Klasse `Exhibition`,
- die Scene, abgebildet durch die Klasse `Scene`,
- das Exhibition Module, abgebildet durch die Klasse `ExhibitionModule`.

Eine virtuelle Ausstellung wird dabei durch genau ein Objekt der Klasse `Exhibition` repräsentiert. Die Scenes und Exhibition Modules einer virtuellen Ausstellung sind in diesem Objekt enthalten. Des Weiteren gilt, dass beim Löschen einer virtuellen Ausstellung die enthaltenen Scenes und Exhibition Modules gleichermaßen gelöscht werden. Die Klasse `Exhibition` steht daher zu den Klassen `Scene` und `ExhibitionModule` durch eine Komposition in Beziehung.

Die Klassen `SceneLink` und `ExhibitionModuleLink` bilden Scene Links und Exhibition Module Links ab. Sie werden dabei durch die Klasse `Link` generalisiert. Des Weiteren wird die Ausgangs-Scene eines Scene Links oder Exhibition Module Links durch das `Scene`-Objekt festgelegt, welches das entsprechende `Link`-Objekt enthält. Die Beziehung zwischen `Scene` und `Link` ist hierbei eine Komposition. Dies ist darin begründet, dass beim Löschen einer Scene die gesamten Scene Links oder Exhibition Module Links, bei denen die Scene Ausgangs-Scene ist, ebenfalls gelöscht werden sollen.

Bei der Verwendung des Altsystems hat sich gezeigt, dass zu Beginn der Erstellung eines Exhibition Modules die Struktur des Exhibition Modules meist noch nicht klar ist. Es ist zunächst notwendig, die Inhalte des Exhibition Modules zu erfassen. Nach der Erfassung können die Inhalte dann strukturiert werden. Beim Altsystem konnte ein Slide oder Branching Point nur innerhalb einer Sequence angelegt werden. Der Slide oder Branching Point war dann genau dieser Sequence zugeordnet. Er war dabei über eine Komposition mit der Sequence verbunden. Die Abhängigkeit der Slides und Branching Points von genau einer Sequence führte bei der Verwendung des Altsystems dazu, dass Sequences mit jeweils einem Slide oder Branching Point erstellt wurden. Die Meta-Sequences wurden dann dazu verwendet, die Reihenfolge der Slides und Branching Points festzulegen. Das Domänenmodell des Prototyps hat daher gegenüber dem Domänenmodell des Altsystems die folgenden Änderungen:

- Slides und Branching Points existieren unabhängig von den Sequences. Sie sind nicht Teil einer Sequence. Die Sequences legen dabei die Reihenfolge der Slides und Branching Points durch Verweise auf diese fest. Slides und Branching Points können somit mehrfach innerhalb eines Exhibition Modules verwendet werden. Ein Slide oder Branching Point, auf den von keiner Sequence verwiesen wird, ist im Navigationsgraph des Exhibition Modules nicht erreichbar.

  Im Domänenmodell des Prototyps wird die beschriebene Änderung zum einen durch die Kompositions-Beziehung zwischen der Klasse `ExhibitionModule` und der Klasse `Slide` deutlich. Die Klasse `Slide` repräsentiert dabei die Slides eines Exhibition Modules. Zum anderen führt diese Änderung zu der Aggregations-Beziehung zwischen der Klasse `Sequence` und der Klasse `Slide`.

- Es gibt keine Meta-Sequences mehr.

Branching Points, abgebildet durch die Klasse `BranchingPoint`, verfügen über die Eigenschaften eines Slides und enthalten darüber hinaus Branching Point Choices. Die Klasse `BranchingPoint` wird daher durch die Klasse `Slide` generalisiert.

Das Layout von Slides und Branching Points wird über Vorlagen, sogenannten *Slide Templates*, gesteuert. Ein Slide Template legt neben der Anordnung von Texten und Bildern die Anzahl der angezeigten Bilder und Texte fest. Einem Slide oder Branching Point können dabei mehr Bilder und Texte hinzugefügt werden als durch das Slide Template angezeigt werden. Im Domänenmodell werden die Slide Templates durch die Klasse `SlideTemplate` repräsentiert. Diese steht in Beziehung zu der Klasse `Slide`. Das System bietet eine Auswahl von Slide Templates, die auf Slides und Branching Points angewandt werden können. Wird ein Slide oder Branching Point gelöscht, wird das verwendete Slide Template jedoch nicht ebenfalls gelöscht. Daher steht die Klasse `Slide` zu der Klasse `Slide Template` in einer Aggregations-Beziehung.

7.5.2. Definition des Metamodells

Zur Definition des Metamodells wurde Java im Kontext von EMF benutzt. Dafür wurde zunächst ein Java Projekt in Eclipse erstellt. Entsprechend der Projektbezeichnung JAT wurde das Projekt `JAT` genannt. Mit Hilfe von EMF wurden daraufhin die folgenden Arbeitsschritte durchgeführt:

- Definition des Metamodells mittels Java-Interfaces
- Erzeugung des Ecore-Modells
- Erzeugung der Java-Klassen für die Metamodellelemente und deren Adapter

Für die Definition des Metamodells wurden zunächst Interfaces implementiert. Dabei entspricht ein Interface einem Metamodellelement. Des Weiteren wird ein Attribut eines Metamodellelements durch eine entsprechende getter-Methode definiert. Die Methode wird hierbei mit `@model` annotiert. Die Annotation muss sich innerhalb eines JavaDoc-Kommentares befinden. Listing 7-1 zeigt ausschnittsweise das annotierte Java-Interface für das Metamodellelement `Exhibition`. Der Parameter `containment=''true''` in Zeile 14 legt dabei fest, dass es sich bei der Beziehung zwischen dem Metamodellelement und dem Attribut um eine Komposition handelt.

```
1  /**
2   * @model
3   */
4  public String getTitle();
5
6  /* ... */
7
8  /**
9   * @model containment="true"
10  */
11 public EList<Scene> getScenes();
```

Listing 7-1 Ausschnitt eines Interface zur Definition des Metamodells

Im nächsten Schritt wurde das Ecore-Modell automatisch durch EMF erzeugt. Die generierte Ecore-Datei befindet sich in Anhang D.2.1. Für die Bearbeitung dieses Ecore-Modells stellt EMF einen Editor zur Verfügung. Ein Screenshot des Editors zeigt Anhang D.2.2.

Nach der Erzeugung des Ecore-Modells wurden die Java-Klassen für die Metamodellelemente und die Adapter für diese Klassen generiert. Für die Generierung benötigt EMF neben dem Ecore-Modell ein weiteres Modell, das *Genmodel*. Dieses enthält hierbei Informationen wie beispielsweise Paketnamen, Pfadangaben oder die zu verwendende Java Compiler Version. Bei der Generierung des Ecore-Modells aus den definierten Interfaces wird das Genmodel ebenfalls automatisch erzeugt. Änderungen am Genmodel bezüglich spezifischer Eigenschaften der zu generierenden Java-Klassen können dann manuell vorgenommen werden. Anhang D.2.3 zeigt das Genmodel von EMF für das definierte Metamodell.

Bei der Generierung durch EMF wurden zum einen die Java-Klassen, die die Metamodellelemente repräsentieren, erstellt. Diese implementieren jeweils das entsprechende, zuvor definierte Interface. Zum anderen wurden einige Hilfsklassen, wie beispielsweise eine Klasse zum Erzeugen von Instanzen der Metamodellelemente[7], und ein zweites Plugin erzeugt. Das generierte Plugin wurde mit Hilfe von EMF.Edit erstellt. Es enthält die Adapter für die Java-Klassen der Metamodellelemente.

7.6. Entwicklung des GMF-Modellierungstools

Die Entwicklung eines GMF-Modellierungstools setzt ein mit EMF erzeugtes Metamodell und die dafür generierten Java-Klassen voraus. Darauf aufbauend werden mittels GMF drei weitere für die Erzeugung eines Modellierungstools notwendige Modelle erstellt: ein *Graphical Definition Model*, ein *Tooling Definition Model* und ein *Mapping Model*. Das Graphical Definition Model enthält dabei graphische Informationen zu den Modellelementen, die im Modellierungstool dargestellt werden. Die graphischen Informationen werden jedoch unabhängig von den Metamodellelementen definiert. Ein Graphical Definition Model kann so für verschiedene Metamodelle verwendet werden. Des Weiteren wird das Tooling Definition Model für die Definition von beispielsweise Kontextmenüs und Toolbars benötigt. Das Mapping Model schließlich verbindet die Informationen aus dem Graphical Definition Model und dem Tooling Definition Model mit den Elementen des Metamodells.

Nachdem alle benötigten Modelle definiert sind, wird aus dem Mapping Model und dem Metamodell mittels GMF ein *Generator Model* erstellt. In diesem Modell können implementierungsspezifische Details festgelegt werden, wie beispielsweise die Menüeinträge, die für das Modellierungstool erzeugt werden. Aus dem Generator Model wird von GMF das Modellierungstool generiert. GMF folgt somit bei der Erstellung eines Modellierungstools selbst dem

[7] Vgl. das Entwurfsmuster “Abstract Factory” bei Gamma et al. ([Gam94]).

Ansatz der modellgetriebenen Softwareentwicklung. Abbildung 7-4 gibt einen Überblick über den Erstellungsprozess eines Modellierungstools mit GMF. ([Ing07])

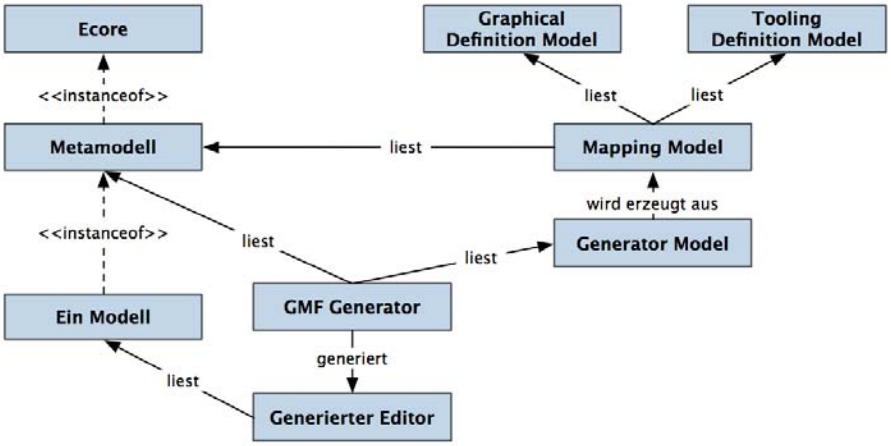


Abb. 7-4. Erstellung eines Modellierungstools mit GMF, Quelle: angelehnt an ([Sta07], S. 106)

7.6.1. Entwicklung einer konkreten Syntax der DSL

Im Rahmen der Entwicklung des Modellierungstools muss neben dem Metamodell die konkrete Syntax der DSL entwickelt werden. Soweit möglich soll die Erstellung eines Modells für eine virtuelle Ausstellung dem Prinzip *What You See Is What You Get* (WYSIWYG) folgen. Das bedeutet, dass die graphische Darstellung der Modellelemente so weit wie möglich der Darstellung der Elemente in den durch die Ausstellungssoftware erzeugten Webseiten für eine virtuelle Ausstellung entspricht. Im Rahmen der Entwicklung einer konkreten Syntax der DSL wurden dabei die folgenden Punkte ausgearbeitet:

- Darstellung einer Virtual Exhibition
- Darstellung von Scenes
- Darstellung von Scene Links und Exhibition Module Links
- Darstellung von Exhibition Modules
- Darstellung von Slides und Branching Points
- Darstellung von Sequences
- Darstellung von Bildern und Texten

Eine Übersicht über die für die konkrete Syntax entwickelten graphischen Elemente befindet sich in Anhang D.3.2.

Für die Definition der graphischen Darstellung der Modellelemente mittels des Graphical Definition Model gibt es zwei Möglichkeiten bei GMF. Zum einen können unterschiedliche geometrische Figuren, die GMF bereitstellt, wie beispielsweise Rechtecke oder Kreise, einzeln oder in Kombination miteinander, zur Darstellung genutzt werden. Zum anderen kann der Entwickler eigene Java-Klassen schreiben, die für die Darstellung verwendet werden. In Anhang D.3.1 befindet sich ein Screenshot des Graphical Definition Models für das zu entwickelnde Modellierungstool.

GEF nutzt die Java-Bibliothek *Draw2d* zur graphischen Darstellung von Modellelementen. Für die Umsetzung der konkreten Syntax der DSL wird bei GMF daher Draw2d verwendet. Draw2d stellt unterschiedliche graphische Komponenten, sogenannte *Figures*, zur Verfügung. Figures können miteinander kombiniert werden, indem sie geschachtelt werden. Das bedeutet, dass eine Figure selbst wieder Figures enthalten kann.

Draw2d setzt auf dem *Standard Widget Toolkit* (SWT) auf. SWT ist eine Klassenbibliothek von IBM, die es ermöglicht, Benutzeroberflächen zu entwickeln, die dem *Look and Feel* des verwendeten Betriebssystems entsprechen. Dafür stellt SWT unterschiedliche *Widgets* zur Verfügung. Ein Widget ist eine Komponente einer Benutzeroberfläche, wie beispielsweise ein Button oder ein Textfeld.[8] Draw2d verwendet SWT, indem es zur Darstellung der Elemente eine `SWT Canvas` nutzt. Eine `SWT Canvas` ist eine Klasse von SWT. Sie kann andere SWT-Objekte enthalten oder als Zeichenfläche dienen, auf die mit geeigneten Operationen gezeichnet werden kann ([Hol04], Kap. 7). Eine Klasse von Draw2d, das `LightweightSystem`, verbindet die Figures mit der genutzten `SWT Canvas`. Abbildung 7-5 zeigt eine schematische Abbildung der Funktionsweise von Draw2d. ([Ecl07a])

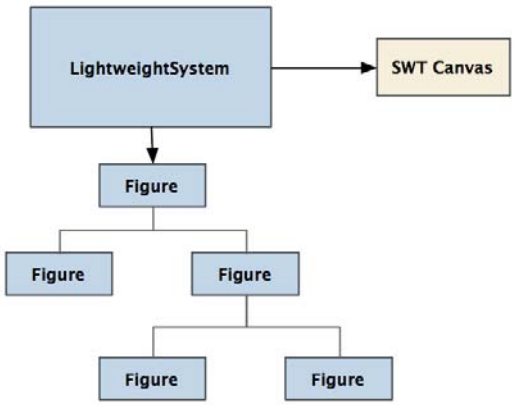


Abb. 7-5. Draw2d, Quelle: angelehnt an ([Ecl07a])

Bei der Entwicklung eines GMF-Modellierungstools wird aus den Elementen des Metamodells ein *Root-Element* ausgewählt. Das Root-Element hat keine graphische Darstellung. Im

[8] Mehr Informationen zu SWT siehe ([Hol04]).

erzeugten Modellierungstool wird es durch die *Zeichenfläche* repräsentiert. Die Zeichenfläche ist der Teil des Modellierungstools, der der Modellierung dient. Auf der Zeichenfläche können hierbei Modellelemente erstellt und bearbeitet werden. Ein Modellelement kann bearbeitet werden, indem es beispielsweise vergrößert oder auf der Zeichenfläche verschoben wird. Im Metamodell des Prototyps ist das Root-Element die `Exhibition`. Für die Virtual Exhibition wird daher keine weitere graphische Darstellung benötigt.

Die graphische Darstellung von Scenes wurde mittels Java-Klassen umgesetzt. Hierfür wurde eine Klasse implementiert, die von einer speziell für die Darstellung von Bildern konzipierten Klasse von Draw2d erbt. Eine Scene wird bei der Modellierung durch das für die Scene gewählte Hintergrundbild repräsentiert. Am oberen Rand des Elements befindet sich darüber hinaus ein Balken, in dem der Titel der Scene angezeigt wird.

Scene Links und Exhibition Module Links wurden mittels einer durch GMF bereitgestellten geometrischen Figure umgesetzt. Hierfür wurde ein Rechteck gewählt, da die Scene Links und Exhibition Module Links in den Webseiten einer virtuellen Ausstellung durch schwarz-transparente Rechtecke dargestellt werden, die den Titel des entsprechenden Links enthalten. Bei der Modellierung können Scene Links und Exhibition Module Links auf dem Bild einer Scene hinzugefügt werden. Des Weiteren können sie innerhalb des Bildes verschoben oder in ihrer Größe verändert werden. Gemäß des WYSIWYG-Prinzips wird dabei die Darstellung der Scene Links und Exhibition Module Links in den generierten Webseiten entsprechend der Positionierung und Größe der Scene Links und Exhibition Module Links im Modell erstellt. Abbildung 7-6 stellt diese Anwendung des WYSIWYG-Prinzips dar. Der linke Teil der Abbildung zeigt dabei Darstellung einer Scene, eines Scene Links und eines Exhibition Module Links im Modell. Der rechte Teil hingegen zeigt die Darstellung der Elemente in der für die Scene generierten Webseite.

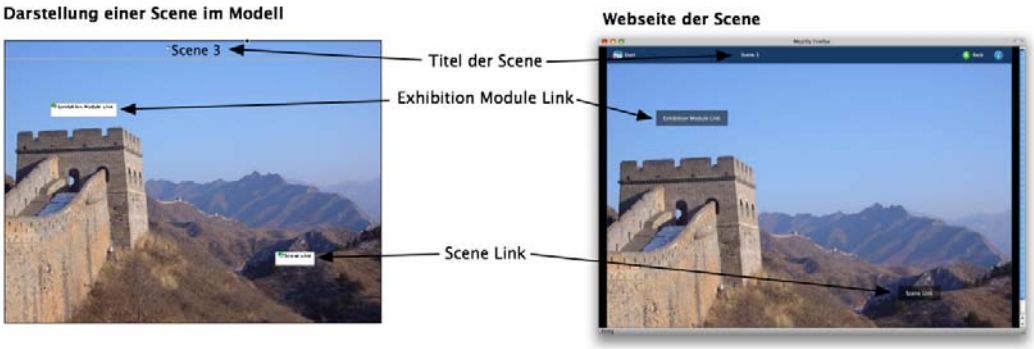


Abb. 7-6. Graphische Darstellung von Scenes, Scene Links und Exhibition Module Links

Die End-Scene für einen Scene Link oder das Exhibition Module für einen Exhibition Module Link wird durch das Erstellen einer Verbindung festgelegt. Die Verbindung wird dabei durch

eine Linie, zwischen dem Rechteck, welches den Scene Link oder Exhibition Module Link abbildet, und dem entsprechenden Modellelement repräsentiert. Diese Art der Repräsentation wurde gewählt, um einen schnellen Überblick über die Navigation durch eine virtuelle Ausstellung zu ermöglichen. In Anhang D.3.3 befindet sich ein Screenshot des Modellierungstools, in dem mehrere Scenes miteinander durch Scene Links verbunden sind.

Für die Exhibition Modules wurde das Symbol eines Netzwerks gewählt. Dieses Symbol stellt das Netzwerk von Sequences eines Exhibition Modules dar. In Anhang D.3.2 befindet sich eine Abbildung der Darstellung eines Exhibition Modules.

Die Darstellung von Slides und Branching Points in den Webseiten einer virtuellen Ausstellung ist annähernd dieselbe. Einziger Unterschied sind die durch HTML-Links dargestellten Branching Point Choices eines Branching Points. Es wurde daher entschieden, dass Slides und Branching Points im Modell, abgesehen von den Branching Point Choices, ebenfalls auf dieselbe Art dargestellt werden.

Die Darstellung eines Slides setzt sich aus drei vertikal angeordneten Bereichen zusammen. Im obersten Bereich wird eine Vorschau der Webseite, die für den Slide generiert wird, angezeigt. Dies dient der Einhaltung des WYSIWYG-Prinzips. Darunter befindet sich jeweils ein Bereich, in dem die dem Slide hinzugefügten Bilder beziehungsweise Texte angezeigt werden. Für die Darstellung von Bildern und Texten wird dabei das entsprechende Bild oder der entsprechende Text verwendet. Branching Points werden auf dieselbe Art wie Slides dargestellt. Sie verfügen jedoch über einen zusätzlichen Bereich, in dem die Branching Point Choices angezeigt werden. Der mehrteilige Aufbau eines Slides oder Branching Points dient dazu, einen Überblick über die hinzugefügten Elemente zu geben. Es ist beispielsweise möglich, einem Slide zwei Bilder hinzuzufügen, gleichzeitig aber ein Slide Template auszuwählen, welches nur ein Bild darstellt. Der Bereich, in dem die Bilder angezeigt werden, ist daher notwendig, um eine Übersicht über alle hinzugefügten Bilder geben zu können. Abbildung 7-7 zeigt die Darstellung eines Slides, der zwei Bilder und einen Text enthält.

Abb. 7-7. Graphische Darstellung eines Slides

Die Darstellung von Slides und Branching Points wurde mit Hilfe der geometrischen Figuren von GMF umgesetzt. Für die Darstellung wurde dabei ein Rechteck gewählt. Diesem Rechteck werden bei der Modellierung weitere Figures, beispielsweise zur Darstellung der dem Slide zugeordneten Bilder, hinzugefügt.

Für die Darstellung von Sequences wurde ebenfalls ein Rechteck gewählt. Dieses Rechteck enthält eine Liste der Titel der Slides und Branching Points, die die Sequence enthält. Die Liste ist in der Reihenfolge sortiert, die die Sequence für die Slides und Branching Points festlegt. Bei der Entwicklung der Darstellung von Sequences wurde die Möglichkeit durchdacht, die Verweise einer Sequence auf Slides und Branching Points, ähnlich wie bei Scene Links, über Linien darzustellen. Ein Exhibition Module kann jedoch eine sehr hohe Anzahl von Slides, Branching Points und Sequences enthalten. Exhibition Modules in der virtuellen Einsteinausstellung enthielten zum Teil bis zu 50 Slides und Branching Points. Würden die Verweise der Sequences durch Linien dargestellt werden, wäre die Struktur eines Exhibition Modules nicht mehr zu erkennen. Eine übersichtliche Anordnung von Slides, Branching Points und Sequences wäre dabei nicht möglich.

7.6.2. Generierung des Modellierungstools

Für die Generierung eines GMF-Modellierungstools muss neben dem Graphical Definition Model das Tooling Definition Model erstellt werden. In diesem Modell wird unter anderem festgelegt, welche Funktionen in der *Palette* dargestellt werden. Die Palette befindet sich neben der Zeichenfläche für die Modellierung. Sie dient der Erstellung der Modellelemente. Abbildung 7-8 zeigt die Palette des entwickelten Modellierungstools. Das Tooling Definition Model des Prototyps befindet sich in Anhang D.3.4.

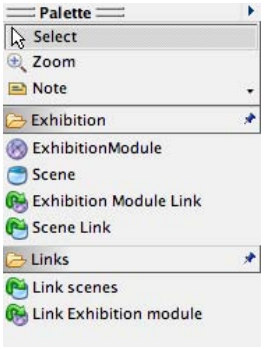


Abb. 7-8. Palette des GMF-Modellierungstools

GMF erzeugt automatisch drei Funktionen für eine Palette. In die Palette wird zunächst ein durch einen weißen Pfeil repräsentiertes Auswahlwerkzeug eingefügt. Dieses Auswahlwerkzeug dient dazu, die Elemente des Modells auszuwählen, um sie beispielsweise zu bearbeiten oder zu bewegen. Des Weiteren wird der Palette eine Zoom-Funktion hinzugefügt, mit der die Ansicht des Modells vergrößert und verkleinert werden kann. Als drittes wird in die Palette eine Funktion zum Erzeugen von Notizen eingefügt. Diese Notizen entsprechen den Notizen in der UML.

Nachdem das Tooling Definition Model und das Graphical Definition Model definiert wurden, ist der nächste Schritt in der Entwicklung eines GMF-Modellierungstools die Erstellung des Mapping Models. Das Mapping Model legt beispielsweise fest, welches Modellelement durch welche Darstellung, die im Graphical Definition Model festgelegt ist, repräsentiert wird. Des Weiteren gibt es an durch welche Funktion der Palette, die im Tooling Definition Modell definiert ist,

ein Modellelement erzeugt werden kann. Abbildung 7-9 zeigt einen Ausschnitt des Mapping Models des Prototyps. In diesem Ausschnitt sind die folgenden drei Elemente eines Mapping Models enthalten: ein `Canvas Mapping`, eine `Top Node Reference` und ein `Node Mapping`.

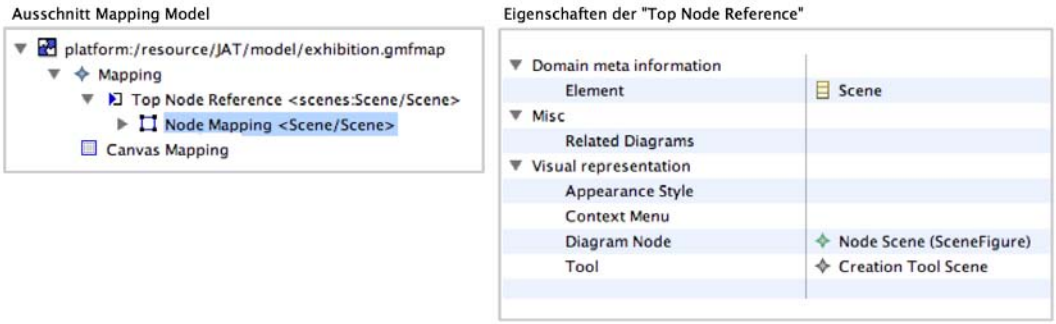


Abb. 7-9. Ausschnitt aus dem Mapping Model des Prototyps

Mit Hilfe des Elements `Canvas Mapping` wird festgelegt, welches Element des Metamodells als Root-Element verwendet wird. Eine `Top Node Reference` wird für Modellelemente verwendet, die direkt auf der Zeichenfläche des Modellierungstools erzeugt werden können. Das Element `Node Mapping` schließlich legt das Metamodellelement für die `Top Node Reference`, dessen Darstellung und die Funktion der Palette zur Erzeugung des Modellelements fest. Dieses Element verbindet somit das Metamodell mit dem Graphical Definition Model und dem Tooling Definition Model. Im rechten Teil von Abbildung 7-9 sind die Eigenschaften des `Node Mapping` zu sehen. Die Eigenschaft `Element` legt dabei das Metamodellelement fest, die Eigenschaft `Diagram Node` die graphische Darstellung des Modellelements und die Eigenschaft `Tool` die Funktion zur Erzeugung des Modellelements.

Aus dem Mapping Model und dem Metamodell wurde das Generator Model für den Prototyp erzeugt. Aus diesem wurde daraufhin ein Eclipse-Projekt generiert. Dieses Eclipse-Projekt enthält die Implementierung des Modellierungstools. In dem Generator Model wurde dabei unter anderem der Name des generierten Eclipse-Projektes, `org.jat.exhibition.diagram`, festgelegt. Dieses Projekt wurde zur Anpassung des Modellierungstools an die definierten Anforderungen um manuell geschriebenen Quelltext erweitert.

7.6.3. Aufbau der Benutzeroberfläche

Die Benutzeroberfläche des als Prototyp entwickelten Modellierungstools hat einen dreiteiligen Aufbau. Dieser setzt sich aus einem Teil für die Modellierung, einem Teil, in dem die Eigenschaften der ausgewählten Modellelemente angezeigt werden, und einer Baumansicht des aktuell im Modellierungstool zu bearbeitenden Modells zusammen. Anhang D.3.5 enthält eine Abbildung, die den Aufbau des Modellierungstools darstellt. Des Weiteren verfügt der Prototyp über zwei unterschiedliche Menüleisten. Eine Menüleiste wird am oberen Bildschirmrand

dargestellt. Diese ist das *Hauptmenü* der Anwendung. Hier befinden sich beispielsweise Funktionen zum Beenden des Programms oder zum Öffnen von Dateien. Das Hauptmenü wird dabei von der Eclipse RCP entsprechend dem Look and Feel des Betriebssystems angepasst. Unter Windows beispielsweise wird diese Menüleiste im Gegensatz zu OS X nicht am oberen Bildschirmrand, sondern am oberen Rand des Anwendungsfensters, angezeigt. Dort wird unter OS X hingegen die zweite Menüleiste des Prototyps angezeigt. Diese wird als *Coolbar* bezeichnet. Die Coolbar wird durch GMF automatisch mit Funktionen, beispielsweise zum Speichern der geöffneten Dateien, gefüllt. Der Hauptmenü- und der Coolbar-Mechanismus werden von der Eclipse RCP bereitgestellt. In Anhang D.3.6 befindet sich ein Screenshot, der den Prototyp mit beiden Menüleisten zeigt.

Der Teil des Modellierungstools, der der Modellierung dient, besteht aus der Zeichenfläche und der Palette. Dieser Teil wird von GMF automatisch generiert. Er wird im Folgenden als *Editor* bezeichnet. Die Darstellung eines Modells oder eines Teils eines Modells im Editor wird dabei als *Diagramm* bezeichnet. Des Weiteren generiert GMF den Teil des Modellierungstools, in dem die Eigenschaften der Modellelemente angezeigt werden. Dieser Teil wird nachfolgend als *Properties-View* bezeichnet. Für die Anzeige der Eigenschaften eines Modellelements im Properties-View muss hierfür das Modellelement mittels des Auswahlwerkzeugs ausgewählt werden. Der dritte Teil der Benutzeroberfläche, die Baumansicht, wurde darüber hinaus speziell für den Prototyp entwickelt.

Bei der Entwicklung des Prototyps wurde deutlich, dass das Diagramm schnell unübersichtlich wird, wenn das Modell mehr als zehn Elemente enthält. Daher wurde entschieden, die Modellierung der Scenes von der Modellierung der Exhibition Modules zu trennen. Für die Erstellung und Bearbeitung einer virtuellen Ausstellung werden daher mehrere Diagramme verwendet. Zunächst gibt es ein Diagramm zur Modellierung der Scenes und der Navigation durch die Scenes. Dieses Diagramm wird im Folgenden als *Hauptdiagramm* bezeichnet. Im Hauptdiagramm können Scenes, Exhibition Modules, Scene Links und Exhibition Module Links erstellt werden. Zur Modellierung der Inhalte eines Exhibition Modules wird darüber hinaus für jedes Exhibition Module ein weiteres Diagramm verwendet, ein *Unterdiagramm*. In diesem Unterdiagramm können Slides, Branching Points und Sequences erstellt werden. Ein Unterdiagramm wird durch einen Doppelklick auf das entsprechende Exhibition Module geöffnet. Der Editor passt sich dabei dem zu bearbeitenden Diagramm an. Das bedeutet zum einen, dass die Palette Diagramm-spezifische Funktionen bereitstellt. Zum anderen können sich die Einträge in der Menüleiste des Programms für das Hauptdiagramm von den Einträgen für das Unterdiagramm unterscheiden.

GMF unterstützt die Erzeugung eines Modellierungstools, das mit Unterdiagrammen arbeitet. Zu diesem Zweck müssen zunächst ein zusätzliches Mapping Model, ein zusätzliches Graphical Definition Model und ein zusätzliches Tooling Definition Model für den Editor des Unterdiagramms erstellt werden. In diesen Modellen werden die Eigenschaften des Unterdiagramms spezifiziert. Im Mapping Model für das Unterdiagramm wird daraufhin für das

`Canvas Mapping` das Metamodellelement festgelegt, welches die im Unterdiagramm zu modellierenden Metamodellelemente enthält. Bei dem entwickelten Prototyp war dieses Element das Metamodellelement `ExhibitionModule`. Im Anschluss daran wird im Mapping Model des Hauptdiagramms dasjenige `Node Mapping` ausgewählt, welches dem im Unterdiagramm als `Canvas Mapping` gesetzten Metamodellelement entspricht. Für dieses `Node Mapping` wird die Eigenschaft `Related Diagrams` auf das `Canvas Mapping` des Unterdiagramms gesetzt. Für das Mapping Model des Unterdiagramms wird ebenfalls ein Generator Model erstellt. Aus diesem Generator Model wird dann ein zweites Eclipse-Projekt erzeugt. Für den Prototyp wurde dabei ein Projekt namens `org.jat.mediastation.diagram` generiert. Dieses Projekt wurde als Plugin in das Eclipse-Projekt für das Hauptdiagramm eingebunden.

Das Eclipse-Projekt für das Hauptdiagramm wird im Folgenden als *Hauptprojekt* des Prototyps bezeichnet. In diesem Projekt werden die Einstellungen für die Eclipse RCP-Anwendung wie beispielsweise die Anordnung von Editor, Properties-View und Baumansicht, vorgenommen. Das Hauptprojekt verwendet neben dem Plugin für das Unterdiagramm weitere speziell für den Prototyp entwickelte Plugins, die nachfolgend aufgeführt werden.

- `org.jat.filehandler`

  Dieses Plugin unterstützt den projektübergreifenden Umgang mit Dateien.

- `org.jat.templates`

  Dieses Plugin verwaltet die Templates des Modellierungstools wie beispielsweise die Slide Templates, die für die Slides und Branching Points verwendet werden.

- `org.jat.workspace`

  Für die Erstellung einer virtuellen Ausstellung wird ein *Projekt-Verzeichnis* angelegt. In diesem Verzeichnis werden die verwendeten Dateien, wie beispielsweise in der virtuellen Ausstellung verwendete Bilder, gespeichert. Darüber hinaus gibt es unterschiedliche temporäre Dateien, die in diesem Verzeichnis gespeichert werden. Dieses Plugin dient dabei der Verwaltung des Projekt-Verzeichnisses.

7.6.4. Validierung des Modells

Eine virtuelle Ausstellung besteht im Durchschnitt aus 20 bis 30 Scenes und etwa 10 bis 20 Exhibition Modules. Die Exhibition Modules enthalten dabei unterschiedlich viele Slides, Branching Points und Sequences. Je mehr Elemente das Modell einer virtuellen Ausstellung enthält, desto eher kommt es zu Modellierungsfehlern. Modellierungsfehler können beispielsweise Scenes sein, die nicht über einen Scene Link mit einer anderen Scene verbunden sind, oder Slides, auf die in keiner Sequence verwiesen wird. Solche Fehler führen dazu, dass es beispielsweise Elemente im Modell gibt, die in der Website der virtuellen Ausstellung nicht

verwendet werden. Für den Prototyp wurde daher eine Funktionalität zur Validierung von Modellen entwickelt.

Die Validierung eines Modells wird beim Prototyp durch den Anwender veranlasst. Die Ergebnisse der Validierung werden dabei in einer tabellarischen Übersicht angezeigt. Zusätzlich wird jedes Modellelement, welches gegen einen Constraint verstößt, durch ein Icon gekennzeichnet. Anhang D.3.7 zeigt einen Screenshot des Prototyps nach der Validierung des Modells. Die Validierungsergebnisse sind in drei Kategorien eingeteilt:

- **Fehler**

  Validierungsergebnisse, die als Fehler eingestuft werden, kennzeichnen Modellierungsfehler, die in der generierten Website dazu führen, dass ein Besucher der Website Probleme feststellt. Solch ein Problem kann beispielsweise ein Exhibition Module Link sein, für den kein Exhibition Module angegeben wurde. Für den Exhibition Module Link wird hierbei ein HTML-Link generiert, dessen Anklicken keinen Effekt hat.

- **Warnungen**

  Validierungsergebnisse, die als Warnungen eingestuft werden, kennzeichnen Modellierungsfehler, die jedoch in den generierten Webseiten zu keinen Problemen führen, die von Besuchern der Website wahrgenommen werden. Warnungen dienen dazu, den Modellierer darauf aufmerksam zu machen, dass das Modell Mängel aufweist, wie beispielsweise Slides, auf die von keiner Sequence verwiesen wird.

- **Informationen**

  Validierungsergebnisse, die als Informationen eingestuft werden, kennzeichnen Modellierungsfehler, die zu keinen Problemen im Modell oder in der generierten Website führen. Informationen dienen ausschließlich dazu, den Modellierer auf Eigenschaften des Modells, die möglicherweise ungewollt sind, aufmerksam zu machen.

Die Kategorisierung der Validierungsergebnisse wurde auf diese Art vorgenommen, um den Modellierer bei der Erstellung eines Modells so wenig wie möglich einzuschränken. Es gibt beispielsweise Modellierungsfehler die vom Modellierer gewollt sind. Eine erste veröffentlichte Version einer virtuellen Ausstellung kann zum Beispiel noch unvollständig sein. Das Modell wird zu einer zweiten Version ausgearbeitet. Die erste Version des Modells kann dabei bereits Inhalte, zum Beispiel in Form von Slides, umfassen, die jedoch noch nicht veröffentlicht werden. Solche Slides sind zu diesem Zweck in keiner Sequence enthalten und werden daher bei der Validierung durch eine Warnung gekennzeichnet.

GMF unterstützt die Modellvalidierung durch die automatische Generierung eines Validierungsmechanismus. Hierfür werden im Mapping Model sogenannte `Audit Rules` definiert. Eine `Audit Rule` beinhaltet einen Constraint. Dieser Constraint kann mit OCL, mit Java oder mit *Regulären Ausdrücken* formuliert sein. Für die Definition der Constraints des Prototyps wurde Java gewählt. Bei der Generierung des Modellierungstools durch GMF wurde

dann ein Klasse erzeugt, die für jede `Audit Rule` eine Methode bereitstellt. Diese Methoden wurden entsprechend der Constraints manuell implementiert. Listing 7-2 zeigt eine solche Methode zur Überprüfung, ob für einen Scene Link oder einen Exhibition Module Link eine End-Scene beziehungsweise ein Exhibition Module festgelegt wurde.

```
1 private static java.lang.Boolean hasTarget(Link self) {
2    if (self instanceof SceneLink)
3       if (((SceneLink) self).getSceneLinkTarget() != null)
4          return true;
5       if (self instanceof ExhibitionModuleLink)
6          if (((ExhibitionModuleLink) self).getExhibitionModuleTarget() !=
               null)
7             return true;
8    return false;
9 }
```

Listing 7-2 Definition eines Constraints

7.6.5. Entwicklung von Slide Templates

Eine spätere Version der Ausstellungssoftware soll die Verwaltung der Slide Templates ermöglichen. Dazu gehört unter anderem das Hinzufügen und Bearbeiten von Slide Templates. Die Template-Sprache der Slide Templates soll daher möglichst einfach erlernbar sein. Im Hinblick auf das Format der durch den Codegenerator erzeugten Dateien wird XHTML in Verbindung mit CSS zur Definition der Slide Templates gewählt. XHTML ist eine Neuimplementierung von HTML auf Basis von XML. Der Vorteil von XHTML-Dokumenten ist dabei, dass sie mit XML-Werkzeugen angezeigt und bearbeitet werden können ([Pem00]). Für die Erstellung eines Slide Templates wird eine Untermenge der XHTML-Elemente verwendet.

Für die Definition eines Slide Templates werden zwei Dateien benötigt. In der ersten Datei wird unter Verwendung des XHTML-Elements `div` das Grundgerüst eines Slide Templates definiert. Diese Datei ist das *JAT Page Template*. In der zweiten Datei wird durch CSS-Angaben die Positionierung der `div`-Elemente der ersten Datei festgelegt. Ein JAT Page Template muss dabei einen XHTML-Kommentar enthalten. In diesem sind eine Id, ein Titel, eine Beschreibung und der Name der zugehörigen CSS-Datei angegeben. Des Weiteren haben JAT Page Templates die Dateiendung `.jatpt` und befinden sich in einem speziellen Verzeichnis innerhalb des Plugins `org.jat.templates`. Die zu den JAT Page Templates gehörenden CSS-Dateien befinden sich ebenfalls in einem speziellen Verzeichnis dieses Plugins. Die Quelltexte eines JAT Page Templates und der dazu gehörenden CSS-Datei befinden sich in Anhang D.3.8.

Die so definierten Slide Templates werden dem Anwender bei der Bearbeitung eines Slides oder Branching Points im Prototyp zur Verfügung gestellt. Die in einem JAT Page Template

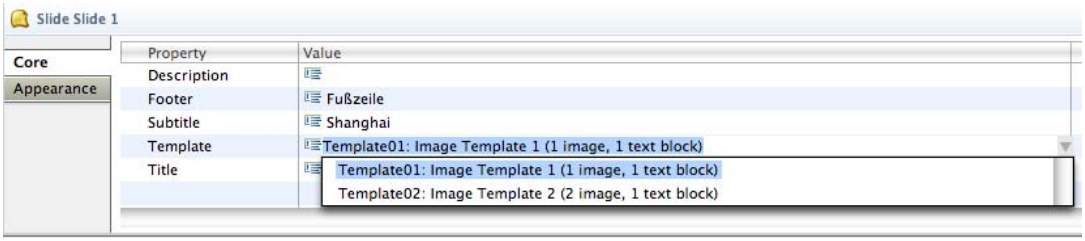


Abb. 7-10. Klappmenü zur Auswahl des Slide Templates

angegebene Beschreibung erscheint dabei in einem Klappmenü im Properties-View des Modellierungstools. Abbildung 7-10 zeigt die Eigenschaften eines Slides und das Klappmenü zur Auswahl des Slide Templates.

7.6.6. Funktionsweise des Modellierungstools

Der Arbeitsbereich des Prototyps innerhalb des Filesystems, im Folgenden auch als *Workspace* bezeichnet, ist das Projekt-Verzeichnis. In diesem Projekt-Verzeichnis werden hierbei die gesamten Dateien gespeichert, die für eine mit dem Prototyp erstellte virtuelle Ausstellung benötigt werden. Im Modell gespeicherte Pfade, wie beispielsweise die Pfadangaben für die verwendeten Bilder, sind relativ bezogen auf den Workspace. Der Grund hierfür liegt darin, dass der Austausch von virtuellen Ausstellungen zwischen unterschiedlichen Rechnern ermöglicht werden soll. Für die Bearbeitung eines Modells auf einem anderen Rechner müssen lediglich das Projekt-Verzeichnis und das Modellierungstool auf diesem Rechner zur Verfügung stehen. Damit soll die Möglichkeit geschaffen werden, dass mehrere Modellierer im Wechsel an einem Modell arbeiten können.

Bei der Erstellung einer neuen virtuellen Ausstellung wird zunächst ein Projekt-Verzeichnis angelegt. In diesem Verzeichnis werden die zwei Dateien gespeichert, mit denen ein GMF-Modellierungstool arbeitet. Die zwei unterschiedlichen Dateien werden im Folgenden als *Modell-Datei* und *Diagramm-Datei* bezeichnet. Die Modell-Datei enthält die Informationen über das Modell. Die Diagramm-Datei enthält die Informationen zur graphischen Darstellung der Modellelemente. Das Hauptdiagramm und die Unterdiagramme des Modellierungstools verwenden dieselbe Modell-Datei und dieselbe Diagramm-Datei.

In einem GMF-Modellierungstool können mehrere Editoren geöffnet sein. Jeder geöffnete Editor wird dabei über einen Reiter am oberen Rand des Teils eines GMF-Modellierungstools angezeigt, in dem die Editoren dargestellt werden. Anhang D.3.9 enthält einen Screenshot des Prototyps mit mehreren geöffneten Editoren. Der Prototyp verfügt über zwei Arten von Editoren, einem Editor für das Hauptdiagramm und einem Editor für die Unterdiagramme. Die Editoren können mehrfach geöffnet sein und jeweils ein anderes Modell darstellen.

Bei der Erstellung einer virtuellen Ausstellung sind in der Regel mehrere Editoren geöffnet. Ein Editor stellt dabei das Hauptdiagramm dar. Darüber hinaus ist der Editor für die Unterdiagramme unter Umständen für jedes Exhibition Module geöffnet. Der Editor eines GMF-Modellierungstools verwaltet ein Modell intern über die sogenannte *Editing Domain*. Jeder geöffnete Editor des Prototyps bildet das Modell über eine eigene Editing Domain ab. Die Editing Domains sind hierbei identisch, da das Hauptdiagramm und die Unterdiagramme auf dieselben Dateien zugreifen. Wird in einem Editor das Modell geändert, bemerken die anderen Editoren dies jedoch erst, wenn sie aktiviert, das heißt erneut angezeigt, werden. Dies kann dann zu Problemen führen, wenn in einem Editor die Änderungen am Modell nicht gespeichert wurden. Die Editing Domains der Editoren sind in diesem Fall nicht mehr identisch. Bei der Entwicklung des Prototyps wurde über die Möglichkeit nachgedacht, das Modell automatisch zu speichern, wenn der Benutzer den Editor wechselt. Dies wurde jedoch nicht umgesetzt, um die Speicherung von ungewollten Änderungen zu vermeiden.

Wichtiger Bestandteil einer virtuellen Ausstellung sind Bilder. Bilder werden für Scenes, Slides und Branching Points verwendet. Ein Bild wird beim Hinzufügen zum Modell einer virtuellen Ausstellung in das Projekt-Verzeichnis kopiert. Dieses enthält hierfür zwei Verzeichnisse. In einem Verzeichnis sind die Bilder der Scenes gespeichert. In dem anderen Verzeichnis sind die Bilder, die in den Slides und Branching Points verwendet werden, gespeichert.

Der Prototyp ermöglicht es, die Bilder, die in den Slides und Branching Points verwendet werden, zu skalieren. Ein Bild hat dafür die Eigenschaften `width` und `height`. Die Größe eines Bildes wird dann verändert, wenn beide Eigenschaften nicht Null sind. Das Bild wird dabei so groß wie möglich proportional skaliert, ohne dass die neue Höhe und Breite des Bildes die angegebenen Werte überschreiten. Abbildung 7-11 zeigt das Prinzip, das bei der Skalierung angewandt wird. Das Originalbild bleibt bei der Skalierung unverändert. Es wird lediglich eine skalierte Kopie des Originalbildes erstellt.

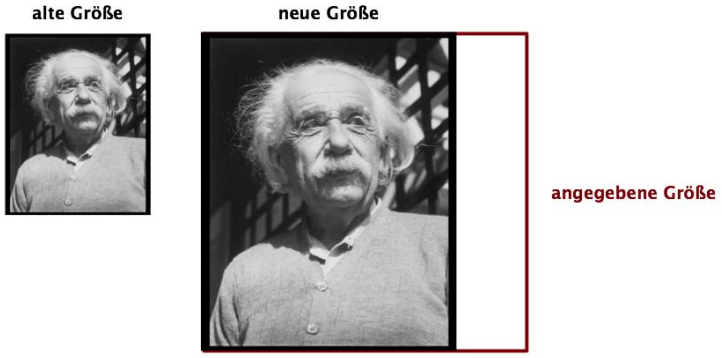


Abb. 7-11. Skalierung eines Bildes

7.7. Entwicklung des Codegenerators

Der im Rahmen der Entwicklung des Prototyps konzipierte und implementierte Codegenerator besteht aus zwei Eclipse-Plugins: `org.jat.generation.engine` und `org.jat.generation`. Er wurde speziell für die Generierung von HTML-Seiten entwickelt. Der Codegenerator arbeitet dabei mittels Templates. Zur Beschreibung der Templates wurde im Rahmen der Entwicklung des Generators eine Template-Sprache konzipiert. Die Template-Engine ist mittels des Plugins `org.jat.generation.engine` umgesetzt. Sie ist für das Parsen der Templates und die Generierung von Dateien anhand der Templates zuständig. Das Plugin `org.jat.generation` übernimmt darauf aufbauende Aufgaben, wie beispielsweise die Zuordnung der Templates zu den unterschiedlichen Elementen des Modells.

7.7.1. Entwicklung einer Template-Sprache

Die Template-Sprache, die für den Prototyp entwickelt wurde, schreibt vor, dass Templates aus beliebigem, statischem Text und Tags zusammengesetzt sind. Tags werden dabei ausgezeichnet, indem sie mit der Zeichenkombination `@%` beginnen und mit `%@` enden. Die Zeichen "@" und "%" wurden für die Auszeichnung ausgewählt, da sie in Kombination miteinander sehr selten innerhalb einer HTML-Seite vorkommen. Eine Suchanfrage bei *Google* nach dieser Zeichenkombination lieferte beispielsweise keine Ergebnisse. Der Name des Tags, der zwischen `@%` und `%@` steht, kann beliebig gewählt werden. Ein Tag hat somit die Form `@%tagname%@`. Anhang D.4.1 enthält ein Template zur Generierung von HTML-Seiten für die Exhibition Modules einer virtuellen Ausstellung.

Die Template-Sprache wurde bewusst generisch entwickelt, um die Möglichkeit zu schaffen, den Codegenerator später um weitere Komponenten zu erweitern. Des Weiteren schränkt die Template-Sprache den Entwickler nur minimal ein. Die Templates können jeden beliebigen Text enthalten und sind bezüglich des Formats, das generiert werden kann, nicht festgelegt. Weitere Komponenten könnten somit beispielsweise statt HTML-Dateien SQL-Skripte erstellen. Mit Hilfe dieser Skripte würden in einer SQL-Datenbank entsprechend der Modellelemente Tabellen erstellt werden. Die im Modell enthaltenen Informationen würden des Weiteren durch diese Skripte in die Tabellen eingefügt werden. Mittels zum Beispiel JSF könnte eine virtuelle Ausstellung dann im Internet präsentiert werden.

Die von dem entwickelten Codegenerator verarbeitbaren Templates werden im Folgenden als *JAT Generation Templates*, kurz *Generation Templates*, bezeichnet. Ein Generation Template hat die Dateiendung `.jatgt`. Des Weiteren muss es sich in dem Verzeichnis `templates` innerhalb der Template-Engine befinden.

7.7.2. Entwicklung der Template-Engine

Die Template-Engine übernimmt innerhalb des Generierungsprozesses die folgenden drei Aufgaben.

- Einlesen des Generation Templates
- Ersetzen der Tags innerhalb eines Templates durch entsprechende Zeichenketten
- Speicherung des daraus resultierenden Textes in einer Datei

Die Template-Engine wird dabei für jede zu generierende Datei separat aufgerufen. Dadurch wird der Prozess der Template-basierten Erzeugung einer Datei gekapselt. Diese Kapselung ermöglicht die Wiederverwendung der Template-Engine. Sie ist somit nicht ausschließlich auf die Verwendung innerhalb des entwickelten Codegenerators festgelegt. Die Template-Engine kann darüber hinaus auch in einem anderen System verwendet werden.

Bei der Initialisierung wird der Template-Engine der Pfad des Verzeichnisses übergeben, in welchem die generierten Dateien gespeichert werden sollen. Dieses Verzeichnis wird im Folgenden als das *Hauptverzeichnis* bezeichnet. Durch die Festlegung eines Hauptverzeichnisses sollen Fehler vermieden werden, die dazu führen, dass die Dateien der Website einer virtuellen Ausstellung an unterschiedlichen Orten im Filesystem gespeichert werden. Mit Hilfe der Template-Engine können dabei Unterverzeichnisse des angegebenen Verzeichnisses erstellt werden. Das Hauptverzeichnis bleibt jedoch dasselbe. Bei Bedarf kann das Hauptverzeichnis über eine Methode der Template-Engine für alle folgenden Generierungen geändert werden.

Beim Aufruf der Funktion zur Generierung einer Datei werden der Template-Engine der Name der zu generierenden Datei, der Name des zu verwendenden Templates und eine *Map* der zu ersetzenden Variablen übergeben. Eine Map ist eine Datenstruktur, die aus Schlüssel-Werte-Paaren aufgebaut ist.[9] Die Schlüssel der Map sind dabei die Tags. Die Werte sind die für die Tags einzusetzenden Zeichenketten. Die Template-Engine liest das entsprechende Template ein, ersetzt die darin vorkommenden Tags durch die entsprechenden Werte in der Map und speichert das Ergebnis unter dem angegebenen Namen.

Bei dem Ersetzungsprozess der Tags wird das Template Zeile für Zeile von der Template-Engine eingelesen. Für jede Zeile werden hierbei die in der Map übergebenen Tags durch die entsprechenden Werte der Reihe nach ersetzt. Tags, die nicht in der Map spezifiziert sind, werden aus dem Text gelöscht. Hierfür werden reguläre Ausdrücke verwendet. Abbildung 7-12 zeigt das Aktivitätsdiagramm des Ersetzungsprozesses der Tags. In Anhang D.4.2 befindet sich der Quelltext zur Ersetzung der Tags.

Die beschriebene Vorgehensweise beim Ersetzungsprozess ermöglicht es, Tags durch Zeichenketten zu ersetzen, die wiederum Tags enthalten. So eingesetzte Tags werden wie Tags behandelt, die im ursprünglichen Template vorhanden sind. Hierbei ist jedoch die Reihenfolge, die

[9] Mehr zu Maps bei Java siehe ([Knu05], Kap. 11.4).

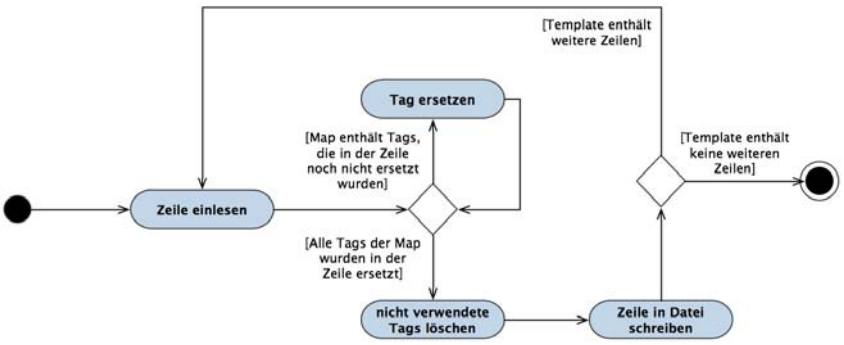


Abb. 7-12. Aktivitätsdiagramm zum Ersetzungsprozess der Tags

die Tags in der Map haben, wichtig. Wird für Tag `A` eine Zeichenkette eingesetzt die Tag `B` enthält, muss Tag `B` in der Reihenfolge der Tags in der Map nach Tag `A` kommen. Nur dann wird Tag `B` mit dem entsprechenden Wert ersetzt.

7.7.3. Funktionsweise des Codegenerators

Das Plugin `org.jat.generation` nutzt die Funktionalität der Template-Engine und baut darauf auf. Es hat dabei die folgenden Aufgaben:

- Dynamisches Erstellen der JAT Generation Templates
- Zusammenstellen der erforderlichen Informationen für den Aufruf der Template-Engine
- Kopieren von Dateien, die für die Website einer virtuellen Ausstellung benötigt werden

Die Generation Templates werden vom Codegenerator zu Beginn des Generierungsprozesses dynamisch erstellt. Dies hat den Grund, dass der Prototyp in einer späteren Entwicklungsphase dahingehend erweitert werden soll, dass die Templates vom Anwender mit Hilfe der Ausstellungssoftware angepasst werden können.

Für die Erzeugung eines Generation Templates wird ein sogenanntes *Master-Template* benötigt. Das Master-Template ist spezifisch für den Typ der Webseite, die erstellt werden soll. Für eine virtuelle Ausstellung gibt es zwei Master-Templates: ein Master-Template für Scenes und ein Master-Template für Exhibition Modules. Ein Master-Template ist hierbei eine XHTML-Datei mit der Dateiendung `.jatmt`. In das Master-Template werden von dem Codegenerator für das Master-Template spezifische Tags eingesetzt. Die einzusetzenden Tags werden dabei über eine XML-Datei festgelegt. Ein Eintrag in der XML-Datei hat die in Listing 7-3 gezeigte Form.

```
<element name="div" id="body">
   <attribute name="style" value="background-image:url(@%backgroundimage%@)
       ; background-repeat:no-repeat"/>
   <content value="@%links%@" />
</element>
```

Listing 7-3 Definition eines einzusetzenden Tags

Das XML-Element `element` definiert hierbei das XHTML-Element des Master-Templates, das modifiziert werden soll. Die Attribute `name` und `id` dienen dazu, das XHTML-Element eindeutig zu bestimmen. Des Weiteren spezifiziert das XML-Element `attribute` ein Attribut, das in das XHTML-Element eingesetzt wird. Das XML-Element `content` hingegen spezifiziert den Inhalt, der in das XHTML-Element eingesetzt wird. Der Codegenerator erzeugt für den in Listing 7-3 gezeigten Eintrag, den in Listing 7-4 dargestellten XHTML-Code.

```
<div id="body" style="background-image:url(@%backgroundimage%@); background
    -repeat:no-repeat">
   @%links%@
</div>
```

Listing 7-4 In Master-Template eingesetzte Tags

Für jedes Master-Template gibt es eine Generator-Klasse, die die für die Generierung benötigten Daten zusammenstellt und aufbereitet. Diese Generator-Klassen werden durch eine abstrakte Klasse, den `AbstractGenerator`, generalisiert. Die Generator-Klassen werden von der Schnittstelle des Plugins `org.jat.generation`, dem `JatGenerationService`, aufgerufen und verwaltet.

Die Schnittstelle der Template-Engine ist die Klasse `JatGenerationEngineFactory`. Diese ist in Anlehnung an das *Abstract Factory*-Entwurfsmuster und das *Strategy*-Pattern entwickelt worden. Im Folgenden wird diese Klasse daher auch als *Factory* bezeichnet. Der `JatGenerationService` erhält über die Factory ein das Interface `JatGeneratorEngine` implementierendes Objekt. Dieses Objekt stellt eine Methode bereit, mit der Dateien eines spezifischen Formats, wie beispielsweise HTML-Dateien, generiert werden können. Das Format der zu generierenden Dateien wird dabei über einen Parameter festgelegt, der der Factory übergeben wird. Bei der Erzeugung der Generator-Klassen reicht der `JatGenerationService` die `JatGeneratorEngine` an die Generator-Klassen weiter. Abbildung 7-13 zeigt das Zusammenwirken der Klassen des Codegenerators.

Die Generator-Klassen erzeugen die Generation Templates und erstellen die Map, die für den Aufruf der Template-Engine benötigt wird. In der Map werden dabei Informationen aus dem Modell und Daten zur Darstellung der Modellelemente gespeichert. Des Weiteren übernehmen die Generator-Klassen das Kopieren der für die Webseite benötigten Dateien wie beispielsweise Bilder und CSS-Dateien. Mit Hilfe der `JatGeneratorEngine` schließlich erzeugen die

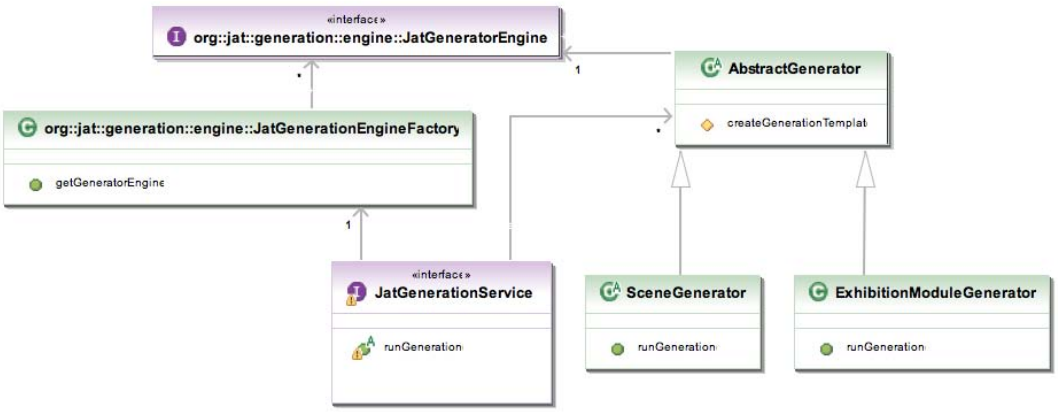


Abb. 7-13. Für die Generierung verantwortliche Klassen

Generator-Klassen die Webseiten für die virtuelle Ausstellung, die durch das Modell abgebildet wird. Die generierten Webseiten werden dabei in einem Verzeichnis `html` im Projekt-Verzeichnis der virtuellen Ausstellung gespeichert.

7.8. Verwendete Entwurfsmuster

Bei der Entwicklung des Prototyps wurden hauptsächlich das *Plugin*-Pattern und das *Singleton*-Pattern umgesetzt. Darüber hinaus wurden wenige Entwurfsmuster verwendet. Dies ist darin begründet, dass der Hauptteil des Quelltextes des Prototyps durch GMF generiert wurde. Die manuell implementierten Teile des Prototyps erweitern dabei hauptsächlich den generierten Quelltext und setzen somit auf den bereits implementierten Entwurfsmustern auf.

Das Plugin-Konzept von Eclipse basiert auf dem *Plugin*-Pattern, wie es bei Fowler et al. zu finden ist.[10] Gemäß diesem Konzept wurden spezielle Funktionalitäten des Prototyps, wie beispielsweise die Template-Verwaltung oder der Codegenerator, über Plugins zur Verfügung gestellt. Die Plugins trennen dabei einen "öffentlichen Teil" von einem "internen Teil". Im "öffentlichen Teil" des Plugins befinden sich die Klassen, die durch das Plugin zur Verfügung gestellt werden. Der "interne Teil" enthält nicht außerhalb des Plugins zur Verfügung stehende Klassen.

Bei der Umsetzung des Plugins `org.jat.workspace` wurde das Singleton-Pattern angewandt.[11] Beim Singleton-Pattern wird sichergestellt, dass von einer Klasse nicht mehr als ein Objekt existiert. Der Prototyp verwendet das Singleton-Pattern, um von unterschiedlichen Klassen in zum Teil unterschiedlichen Plugins auf denselben Workspace zuzugreifen. Beim ersten Zugriff auf den Workspace wird dabei ein Objekt der Klasse, die das Singleton-Pattern umsetzt,

[10] Siehe hierzu ([Fow02], S. 499ff.)
[11] Vgl. hierzu das Entwurfsmuster "Singleton" bei Gamma et al. ([Gam94], S. 127).

erzeugt. Dieses Objekt hält die notwendigen Informationen über den Workspace. Weitere Zugriffe auf den Workspace erfolgen ausschließlich über dieses Objekt. In Anhang D.5.2 befindet sich ein Auszug des Quelltextes der Klasse, die das Singleton-Pattern umsetzt. Der vollständige Quelltext befindet sich auf der der Diplomarbeit beiliegenden CD.

7.9. Implementierungsdetails

Bei der Entwicklung des Prototyps erwies sich die Umsetzung einiger Details als komplizierter als erwartet. Dies war zum großen Teil bedingt durch GMF. Im Folgenden wird die Entwicklung von drei dieser Details ausführlicher beschrieben.

7.9.1. Baumansicht des Modellierungstools

Die Baumansicht des Modellierungstools, dargestellt in Abbildung 7-14, wurde unter Verwendung eines *Tree Viewers* von *JFace* entwickelt. JFace basiert auf SWT und dient der Entwicklung von graphischen Benutzeroberflächen.[12] Ein Tree Viewer in JFace benötigt einen *Content Provider* und einen *Label Provider*. Der Content Provider verwaltet dabei die Objekte, die durch den Tree Viewer angezeigt werden, und deren Struktur. Der Label Provider hingegen dient der graphischen Darstellung der Objekte in dem Tree Viewer.

Zunächst wurde erwogen, den Content Provider und den Label Provider vollständig neu zu implementieren. Nach eingehender Recherche wurde jedoch entschieden, für die Implementierung des Content Providers und des Label Providers speziell für diesen Zweck konzipierte Klassen zu nutzen, die von EMF bereitgestellt werden. Dafür wurden eigene Klassen entwickelt, die von den EMF-Klassen erben. Durch die Verwendung der Klassen von EMF war die Hauptfunktionalität des Content Providers und des Label Providers bereits umgesetzt. Es wurden lediglich geringfügige Ergänzungen vorgenommen, wie beispielsweise das Hinzufügen einer zusätzlichen Ebene in der Baumansicht, wie in Abbildung 7-14 zu sehen, die die Scenes und Exhibition Modules getrennt voneinander darstellt.

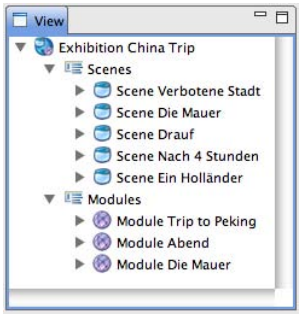


Abb. 7-14. Baumansicht des Modellierungstools

Der nächste Schritt in der Entwicklung der Baumansicht war das Anzeigen der Eigenschaften eines Elements, das in der Baumansicht ausgewählt wurde, im Properties-View des Modellierungstools. Dafür musste zunächst der Tree Viewer bei der *Site* des *Views*, in dem sich der Tree Viewer befindet, registriert werden (siehe Listing 7-5).

```
1 getSite().setSelectionProvider(viewer);
```

Listing 7-5 Registrierung des Tree Viewers an der Site

[12]Mehr Informationen zu JFace und SWT siehe http://wiki.eclipse.org/The_Official_Eclipse_FAQs#JFace.

Ein View ist dabei ein Teil eines Fensters einer mit Eclipse erstellten Anwendung. Abbildung 7-14 zeigt beispielsweise den View, in dem sich die Baumansicht befindet. *«The site is not a visible entity but simply an API mechanism to separate the methods that operate on the view from the methods that operate on controls and services outside the view.»* ([Pea04])[13] Der Quelltext des Content Providers, des Label Providers und des Views der Baumansicht befinden sich auf der der Diplomarbeit beigefügten CD.

Zuletzt wurde in der `plugin.xml` des Hauptprojekts das Objekt, welches im Properties-View angezeigt werden soll, bekannt gemacht. Die `plugin.xml` enthält die Informationen, die für ein Plugin oder eine Eclipse RCP-Anwendung benötigt werden. Für die Bekanntmachung des Objekts, das im Properties-View angezeigt werden soll, wurde die Zeile `<input type=‘‘org.eclipse.emf.ecore.EObject"/>` an entsprechender Stelle in die `plugin.xml` eingefügt. Listing 7-6 zeigt den hierfür relevanten Teil der `plugin.xml` mit dem in Zeile 7 eingefügten XML-Element. Der vollständige Quelltext der `plugin.xml` befindet sich in Anhang D.6.1. Zur Bekanntmachung wurde die Klasse `EObject` gewählt, da alle im Tree Viewer angezeigten Objekte durch diese Klasse generalisiert werden.

```
1 <propertySection
2   id="property.section.domain"
3   tab="property.tab.domain"
4   class="org.jat.exhibition.diagram.sheet.ExhibitionPropertySection">
5     <input type="org.eclipse.gmf.runtime.notation.View"/>
6     <input type="org.eclipse.gef.EditPart"/>
7     <input type="org.eclipse.emf.ecore.EObject"/>
8 </propertySection>
```

Listing 7-6 Anpassung der plugin.xml

7.9.2. Vorschaubilder in Slides und Branching Points

Ein weiteres Detail, dessen Umsetzung mehr Zeit als erwartet in Anspruch nahm, war die Vorschau der generierten Webseiten für Slides und Branching Points. Hierbei war jedoch nicht die Umsetzung selbst zeitaufwendig, sondern die Entwicklung eines Lösungskonzepts für die Problemstellung. Die erste Überlegung diesbezüglich war die Verwendung eines *Browser Widgets*. Ein Browser Widget ist eine von SWT bereitgestellte Komponente, die das Anzeigen von HTML-Dokumenten ermöglicht. Problematisch bei dieser Art der Umsetzung war, dass Draw2d die Verwendung von SWT Widgets nicht zulässt. Es existieren zwar Hilfskonstruktionen, bei denen ein Bild eines Widgets erstellt wird, welches dann in einer Figure eingebunden wird. Für die vorliegende Problemstellung waren diese aber nicht anwendbar.

Eine umfassende Recherche führte zu der Entscheidung, den *XHTML-Renderer Flying Saucer* zu verwenden. Ein XHTML-Renderer wird für die Darstellung von XHTML-Dokumenten

[13] Weitere Informationen zum Aufbau und der Funktionsweise einer Anwendung, die mit Eclipse erstellt wurde, findet sich unter http://wiki.eclipse.org/The_Official_Eclipse_FAQs.

verwendet. Flying Saucer wurde unter der *GNU Lesser General Public License* veröffentlicht und ist in Java implementiert. Er ermöglicht das Rendern von XHTML-Dokumenten, in denen CSS-Angaben verwendet werden. Von der gerenderten XHTML-Seite liefert Flying Saucer dabei unter anderem ein Bild zurück.[14]

Für die Erzeugung des Vorschaubildes für einen Slide oder Branching Point wird zunächst die Webseite generiert. Hierfür wird das Master-Template für Exhibition Modules verwendet und das für den Slide oder Branching Point ausgewählte Jat Page Template. Die erzeugte Webseite wird dabei temporär zwischengespeichert. Mit dem in Listing 7-7 dargestellten Quelltext wird dann durch Flying Saucer ein Bild der Webseite erstellt, das in eine Figure eingebunden wird.

```
1 BufferedImage buff = null;
2 try {
3    buff = ImageRenderer.renderToImage("file:"
4          + templateFile.getAbsolutePath(), folder.getAbsolutePath()
5          + "/tempImage", 1024);
6 } catch (IOException e) {
7    e.printStackTrace();
8 }
```

Listing 7-7 Erzeugung des Vorschaubildes durch Flying Saucer

7.9.3. Erweiterung der Validierung

Der von GMF generierte Validierungsmechanismus ermöglicht die Kennzeichnung eines Modellelements, das gegen einen Constraint verstößt, durch ein Icon. Für den Prototyp sollte jedoch darüber hinaus eine Tabelle implementiert werden, die die gesamten Validierungsergebnisse anzeigt. Zu einem Validierungsergebnis sollten dabei das Modellelement, die Kategorie des Validierungsergebnisses und der Text, der den Modellierungsfehler beschreibt, dargestellt werden. Abbildung 7-15 zeigt die Tabelle der Validierungsergebnisse des Prototyps.

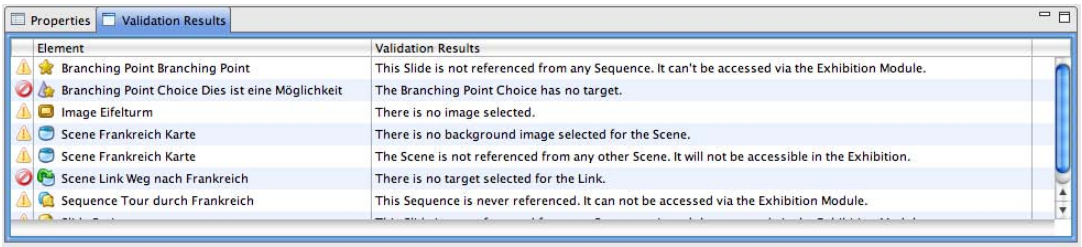


Abb. 7-15. Tabelle der Validierungsergebnisse

Bei dem von GMF generierten Validierungsmechanismus wird ein Validierungsergebnis durch ein Objekt der Klasse `ValidationMarker` repräsentiert. Dieses Objekt enthält die Kategorie

[14]Mehr Informationen zu Flying Saucer siehe https://xhtmlrenderer.dev.java.net/.

des Validierungsergebnisses und den Text, der den Modellierungsfehler beschreibt. Es gibt des Weiteren darüber Auskunft, welchem Metamodellelement das validierte Modellelement entspricht. Das konkrete validierte Modellelement kann jedoch über das Objekt nicht erhalten werden. Für die Tabelle der Validierungsergebnisse wurde jedoch das validierte Modellelement benötigt, um beispielsweise den Titel des Modellelements in der Tabelle anzuzeigen. Für diesen Zweck wurde daher die Klasse `ValidationMarker` um einen zweiten Konstruktor erweitert. Diesem Konstruktor wird unter anderem das validierte Modellelement übergeben. Als Typ des Parameters für das übergebene Modellelement wurde die Klasse `EObject` gewählt, da alle Modellelemente durch diese Klasse generalisiert werden. Listing 7-8 zeigt den hinzugefügten Konstruktor. In Anhang D.6.2 befindet sich der vollständige Quelltext der Klasse.

```
1 public ValidationMarker(String location, String message,
2       int statusSeverity, EObject element) {
3    this.location = location;
4    this.message = message;
5    this.statusSeverity = statusSeverity;
6    this.element = element;
7 }
```

Listing 7-8 Hinzugefügter Konstruktor der Klasse ValidationMarker

Für die Anzeige des Modellelements in der Tabelle der Validierungsergebnisse wurde das Konzept der *Item Providers* von EMF genutzt. Item Provider dienen bei EMF unter anderem dazu, Funktionen für Content Provider und Label Provider bereitzustellen ([Ste03], Kap. 3.2). Die Adapter, die EMF für die Modellelemente generiert, sind zusätzlich auch Item Provider.

Bei der Anzeige eines Modellelements in der Tabelle der Validierungsergebnisse wird die Klasse `DelegatingWrapperItemProvider` von EMF genutzt. Der Label Provider der Tabelle wendet sich dabei an diese Klasse, um den Text und das Bild, die für ein konkretes Element in der Tabelle angezeigt werden, zu erhalten. Der `DelegatingWrapperItemProvider` delegiert die Anfragen an den entsprechenden Item Provider des Modellelements. In Anhang D.6.3 befindet sich der Quelltext des Label Providers. Abbildung 7-16 zeigt ein Sequenzdiagramm zur Arbeitsweise des `DelegatingWrapperItemProvider`.

Der Label Provider übergibt dem `DelegatingWrapperItemProvider`, wie in Abbildung 7-16 zu sehen, bei dessen Initialisierung das Modellelement, welches in der Tabelle der Validierungsergebnisse angezeigt werden soll. Der `DelegatingWrapperItemProvider` wendet sich daraufhin an eine `AdapterFactory`. Diese liefert für das übergebene Modellelement den entsprechenden Item Provider zurück. Eine Anfrage des Label Providers beispielsweise nach dem Text delegiert der `DelegatingWrapperItemProvider` dann an den so erhaltenen Item Provider. Das von dem Item Provider zurückgelieferte Ergebnis wird dabei an den Label Provider zurückgegeben.

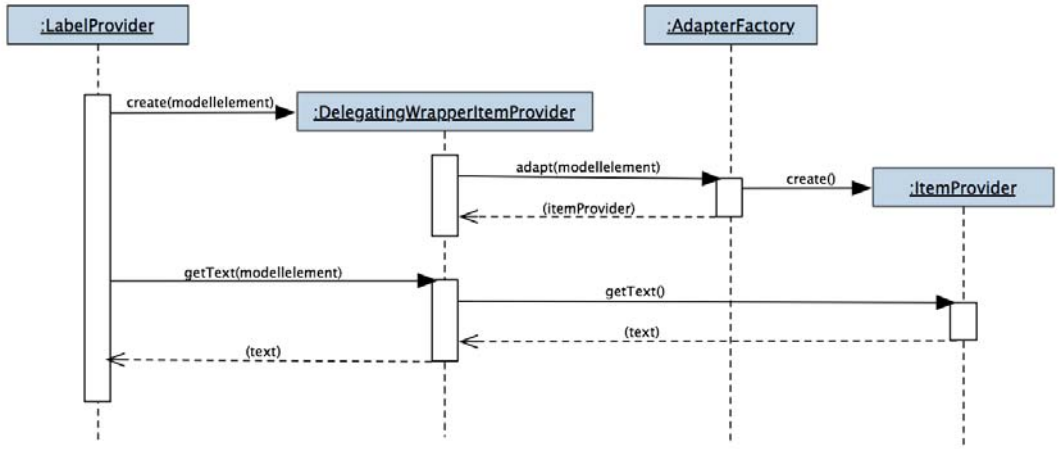


Abb. 7-16. Sequenzdiagramm zur Arbeitsweise des `DelegatingWrapperItemProvider`

7.10. Teststrategien

Zur Sicherung der Softwarequalität wurden im Rahmen dieser Diplomarbeit Anwendertests und Unit-Tests durchgeführt. Die Anwendertests dienten hauptsächlich der Bewertung des Prototyps bezüglich der Bedienbarkeit des Systems. Sie wurden jedoch teilweise auch zur Identifizierung von Softwarefehlern genutzt.

Für die Implementierung der Unit-Tests wurde *JUnit* in der Version 4 verwendet. Der Prototyp wurde jedoch nicht vollständig durch Unit-Tests getestet, da ein großer Teil des Quelltextes generiert wurde. Mittels EMF lässt sich ein Gerüst von JUnit-Tests zum Testen der Metamodell-Klassen generieren. Mit Hilfe dieser Test-Klassen können jedoch nur die Methoden zum Setzen und Abfragen von Attributen getestet werden. Der manuell hinzugefügte Code beschränkt sich weitgehend auf die graphische Benutzeroberfläche, welche sich nur schlecht mit Unit-Tests testen lässt. Der Versuch, die graphische Benutzeroberfläche mit Hilfe des Eclipse Plugins *Test & Performance Tools Platform* (TPTP) zu testen, scheiterte.[15] Der Codegenerator wurde teilweise mit Unit-Tests getestet. Die Testergebnisse befinden sich in Anhang D.7.

[15]Mehr Informationen zu TPTP siehe http://www.eclipse.org/tptp/.

Kapitel 8.

Anwendertests

Zur Durchführung der Anwendertests wurden 7 Testpersonen, die hauptsächlich aus dem Umfeld des MPIWG stammen, ausgewählt. Diese Testpersonen konnten neben der Bewertung der Bedienbarkeit der Ausstellungssoftware auch den Prototyp mit dem Altsystem vergleichen.

Den Anwendertests vorausgegangene Tests bezüglich der Bedienbarkeit des Prototyps hatten ergeben, dass eine Einführung über die Funktionsweise des Prototyps erforderlich ist. Daher wurde den Testpersonen zunächst die Bedienung des Prototyps erläutert. Die Einführung umfasste dabei etwa zehn Minuten. Für den späteren Einsatz der Ausstellungssoftware wird hierfür in einer dem Prototyp folgenden Entwicklungsphase ein Handbuch verfasst werden.

Der für den Prototyp konzipierte Anwendertest umfasste 16 Aufgaben. Bei korrekter Bearbeitung der Aufgaben entstand dabei eine virtuelle Ausstellung, für die in der letzten Aufgabe die Website generiert wurde. Für jede Aufgabe wurde die Bearbeitungszeit gemessen und auf dem Testbogen notiert. In den Anwendertest wurden bewusst Wiederholungen von unterschiedlichen Aktionen eingebaut, wie zum Beispiel das Anlegen einer Scene in zwei unterschiedlichen Aufgaben. Dies sollte die Erlernbarkeit der Software testen. Des Weiteren wurde die Schwierigkeit der Aufgaben mit einem Wert zwischen eins (einfach) und vier (schwer) von den Testpersonen eingeschätzt. Im Anschluss an die Bearbeitung der Aufgaben sollten die Testpersonen darüber hinaus die Ergonomie des Prototyps ebenfalls mit Werten zwischen eins und vier bewerten. In Anhang E.1 befindet sich der Testbogen zum Anwendertest. Die Durchführung der Tests wurde durch die Entwicklerin betreut. Diese stand dabei den Testpersonen für Fragen zur Verfügung. Die Inanspruchnahme von Hilfe durch die Entwicklerin zur Bearbeitung einer Aufgabe wurde hierbei auf dem Testbogen vermerkt.

Für die Bearbeitung der Aufgaben benötigten die Testpersonen im Durchschnitt 28 Minuten. Einige Testpersonen lagen dabei etwa zehn Minuten unter dem Durchschnitt. Dies ist damit zu erklären, dass diese Testpersonen bereits Erfahrung im Umgang mit einer IDE hatten. Die meisten IDEs haben, ähnlich wie der Prototyp, einen mehrteiligen Aufbau mit einem Properties-View und einer Baumansicht zur Navigation. Insbesondere der Properties-View bereitete den Testpersonen Probleme, die keine Erfahrung im Umgang mit IDEs hatten. Das Konzept die Eigenschaften eines Elements über ein zusätzliches Fenster zu bearbeiten,

scheint ohne entsprechende Einarbeitung nicht intuitiv zu sein. Bei Programmen wie *Adobe Photoshop* oder *Adobe Illustrator* beispielsweise werden die Eigenschaften eines Elements im Gegensatz zur Modellierung mittels des Prototyps direkt am Element geändert. Text wird bei diesen Programmen beispielsweise direkt an der Stelle eingegeben, an der er erscheinen soll. Anhang E.2.1 enthält ein Diagramm zur Bearbeitungszeit der Aufgaben.

Abbildung 8-1 zeigt ein Diagramm, in dem die für jede Aufgabe durchschnittlich benötigte Zeit dargestellt ist. Bei Betrachtung des Diagramms fällt dabei zunächst der Maximalwert der abgebildeten Kurve bei Aufgabe fünf auf. In Aufgabe fünf sollte das Unterdiagramm für ein Exhibition Module geöffnet und ein Slide angelegt werden. Dem Slide sollten dabei ein Text und zwei Bilder hinzugefügt werden. Die Testpersonen taten dies im Rahmen des Anwendertests zum ersten Mal, was den maximalen Wert der Kurve erklärt. In Aufgabe sechs wurden daraufhin zwei weitere Slides angelegt. Hierfür war nur noch etwa die Hälfte der für Aufgabe fünf benötigten Zeit erforderlich. Bei Betrachtung des Verlaufs der Kurve kann darüber hinaus

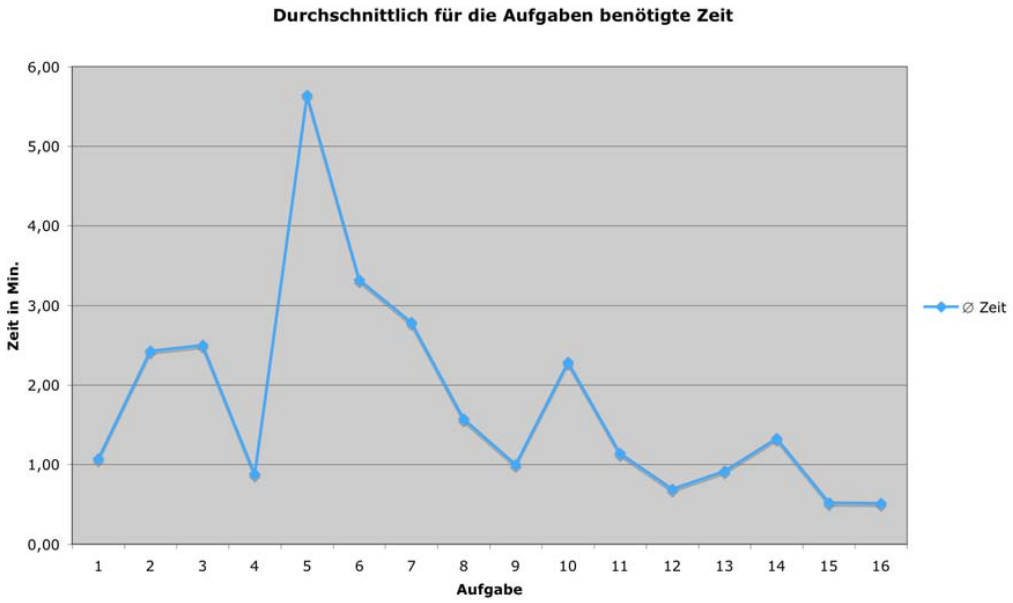


Abb. 8-1. Bearbeitungszeit pro Aufgabe

festgestellt werden, dass die benötigte Zeit für die Aufgaben im Verlauf des Anwendertests tendenziell abnimmt. Dies ist hauptsächlich damit zu erklären, dass die Testpersonen im Umgang mit dem Prototyp gegen Ende der Bearbeitung des Anwendertests geübter wurden als zu Anfang. Hinzu kommt jedoch auch, dass manche Aufgaben, zum Beispiel Aufgabe 15 und 16, darüber hinaus erheblich kürzer waren als andere Aufgaben.

Der Schwierigkeitsgrad der Aufgaben wurde von den Testpersonen fast ausschließlich im Bereich “leicht” eingestuft. Das bedeutet, dass die Werte eins oder zwei vergeben wurden. Nur die Aufgaben zwei und drei empfanden manchen Testpersonen als schwer. Dies ist damit zu

erklären, dass in Aufgabe zwei zum ersten Mal im Rahmen des Anwendertests ein Modellelement mit Hilfe der Palette angelegt wurde. Darüber hinaus wurde in dieser Aufgabe zum ersten Mal der Properties-View verwendet. In Aufgabe drei wurde hingegen ein Scene Link erstellt. Dafür musste mit Hilfe der Palette ein Scene Link erzeugt werden, der dann mit einer anderen Scene verbunden wurde. Insbesondere die Erstellung der Verbindung zwischen Scene Link und Scene bereitete einem Großteil der Testpersonen Schwierigkeiten. Ein Diagramm zur Einschätzung der Schwierigkeit der Aufgaben befindet sich in Anhang E.2.2.

Bei der Bearbeitung des Anwendertests wurde im Durchschnitt für jede Aufgabe von 22% der Testpersonen Hilfe von der betreuenden Entwicklerin benötigt. Die Hilfe bestand meistens darin, der Testperson einen Hinweis zu geben, wie eine Aufgabe zu bearbeiten war oder in welchem Teil des Prototyps, zum Beispiel im Properties-View oder im Editor, die Testperson eine Aktion ausführen sollte. In Anhang E.2.3 befindet sich ein Diagramm zur Hilfe, die die Testpersonen für die Bearbeitung der Aufgaben benötigten. Bei dessen Betrachtung fällt auf, dass insbesondere bei den Aufgaben öfter Hilfe benötigt wurde, für deren Lösung im Gegensatz zu den anderen Aufgaben auch mehr Zeit beansprucht wurde oder die als schwieriger eingestuft wurden.

Die Analyse der Fragen zur Ergonomie des Prototyps ergab, dass die Anwendung von den meisten Testpersonen zwar nur zum Teil als intuitiv, dafür jedoch als sehr schnell erlernbar eingestuft wurde. Abbildung 8-2 zeigt hierzu die Einschätzung der Erlernbarkeit des Prototyps durch die Testpersonen. Die prozentualen Angaben beziehen sich dabei auf die Gesamtheit aller gegebenen Antworten. Auch die Menüeinträge wurden von den meisten Testpersonen dort gefunden, wo sie erwartet wurden. Die Aufteilung der Funktionalität der Anwendung auf ein Hauptmenü am oberen Bildschirmrand und eine weitere Menüleiste am oberen Rand des Anwendungsfensters, wurde von den Testpersonen nicht als negativ empfunden.

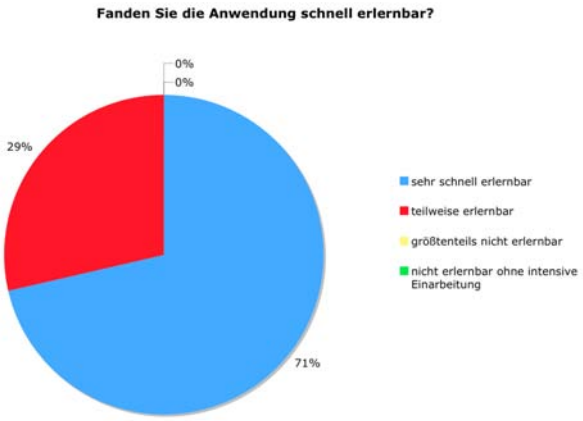


Abb. 8-2. Einschätzung der Erlernbarkeit des Prototyps

Die Symbole und Beschriftungen der Bedienungselemente wurden hingegen von den meisten

Testpersonen als nur teilweise aussagekräftig eingeschätzt. Einige Testpersonen bewerteten die Symbole und Beschriftungen sogar als wenig bis gar nicht aussagekräftig. Anhang E.2.4 enthält weitere Diagramme bezüglich der Beantwortung der Fragen zur Ergonomie des Prototyps.

Kapitel 9.

Ergebnisse

Der Bewertung des Prototyps werden die Ergebnisse der Anwendertests und die Erfahrungen, die während der Entwicklung des Prototyps gemacht wurden, zu Grunde gelegt. Dabei soll entschieden werden, inwieweit der Ansatz der modellgetriebenen Softwareentwicklung unter Verwendung von GMF für die Entwicklung einer Ausstellungssoftware geeignet ist. Im Folgenden werden daher zunächst die Ergebnisse der Anwendertests und die während der Entwicklung gesammelten Erfahrungen ausgewertet. Im Anschluss wird ein Fazit aus dieser Auswertung gezogen.

9.1. Auswertung

Die Funktionsweise des entwickelten Prototyps ist, da er als Eclipse RCP entwickelt wurde, stark von Eclipse geprägt. Insbesondere Anwender, die keine Erfahrung im Umgang mit dieser IDE haben, benötigen eine detaillierte Einweisung in die Bedienung des Prototyps. Konzepte wie beispielsweise der Properties-View sind für diese Anwender dabei nicht intuitiv verständlich. Das entwickelte System ist jedoch schnell erlernbar, wenn der Anwender durch entsprechende Hilfestellungen unterstützt wird. Die schnelle Erlernbarkeit zeigte sich im Rahmen der Anwendertests unter anderem darin, dass die Wiederholung einer Aktion, wie beispielsweise die Erstellung eines Slides, in kürzerer Zeit ausgeführt und als weniger schwer empfunden wurde als die erste Ausführung der Aktion. Für den erfolgreichen Einsatz des entwickelten Systems ist daher ein Handbuch oder zumindest eine Einführung in die Bedienung des Systems erforderlich.

Des Weiteren wurde mittels des Prototyps im Rahmen der Anwendertests eine minimale Version einer virtuellen Ausstellung in durchschnittlich einer halben Stunde erstellt. Hinzu kam die etwa zehnminütige Einführung. Für die Umsetzung von virtuellen Ausstellungen waren dabei neben den erforderlichen Kenntnissen in der Bedienung des Prototyps keine weiteren Kenntnisse notwendig. Im Gegensatz dazu waren für den Einsatz des Altsystems eine umfassende Schulung sowie Kenntnisse mit dem Applikationsserver Zope erforderlich.

Der Prototyp stellt somit in Bezug auf die Erlernbarkeit und Bedienbarkeit gegenüber dem Altsystem eine Verbesserung dar.

Die Anwendertests haben jedoch auch die Schwachstellen des Prototyps aufgezeigt. Die Symbole und die Beschriftungen der Bedienungselemente müssen beispielsweise überarbeitet werden. Des Weiteren muss bei manchen Funktionen die Performance der Anwendung verbessert werden oder der Anwender muss durch einen Fortschrittsbalken über den Status der Ausführung einer Funktion informiert werden. Damit soll vermieden werden, dass der Anwender fälschlicherweise annimmt, die Anwendung sei abgestürzt. Weiterhin sollte zum Beispiel das Verfahren zum Erstellen eines Scene Links überarbeitet werden. Die Anwendertests haben gezeigt, dass die dabei angewandte Vorgehensweise nicht intuitiv verständlich ist. Dies ist teilweise darin begründet, dass die Erstellung eines Scene Links stark von GMF geprägt ist. So kann zum Beispiel nur der obere Rand einer Scene, bedingt durch GMF, mit einem Scene Link verbunden werden.

Bezüglich GMF wurde im Rahmen dieser Diplomarbeit festgestellt, dass dieses Framework die Entwicklung eines Modellierungswerkzeugs auf unterschiedliche Art unterstützt und vereinfacht. Die automatische Generierung des Editors oder des Validierungsmechanismus beispielsweise verringert die Arbeit des Entwicklers zum Teil erheblich. Problematisch bei der Verwendung von GMF ist allerdings, dass das Framework noch einige *Bugs* enthält. Bei dem Prototyp gibt es beispielsweise das Problem, dass nach der Durchführung der Validierung die Zoom-Funktion nicht mehr ausgeführt werden kann. Die Oberfläche friert ein, bei dem Versuch das Diagramm zu vergrößern oder zu verkleinern. Des Weiteren gibt es Eigenschaften eines mit GMF erstellten Modellierungstools, wie beispielsweise die Zoom-Funktion, die der Entwickler nicht beeinflussen kann. Diese Eigenschaften sind im Framework verankert. Darüber hinaus ist bei GMF die eingeschränkte Entwicklung der graphischen konkreten Syntax durch die Verwendung von Draw2d zu bemängeln. Manche graphischen Komponenten, wie beispielsweise das Browser Widget, können so zur Umsetzung einer konkreten Syntax nicht verwendet werden.

Bei der Entwicklung des Codegenerators wurde festgestellt, dass eine Modell-zu-Modell-Transformation vor der Generierung der Webseiten, den Generierungsprozess übersichtlicher und weniger komplex gestalten würde. In dieser M2M-Transformation würden die Modell-Datei und die Diagramm-Datei in ein Modell überführt werden, welches alle Informationen für die Generierung der Webseiten enthält. Des Weiteren verfügt der entwickelte Codegenerator nicht über die Funktionalität, generierten und manuell erstellten Quelltext bei einer erneuten Codegenerierung zusammenzuführen. Die generierten Webseiten sind jedoch auf Grund ihres XHTML-Formats dafür ausgelegt. Der Codegenerator könnte somit dahingehend erweitert werden, dass Protected Regions durch XHTML-Kommentare ausgezeichnet werden.

9.2. Fazit

Grundsätzlich kann zum Prototyp gesagt werden, dass das Konzept der Abbildung einer virtuellen Ausstellung durch ein Modell den Testpersonen keine Schwierigkeiten bereitete. Die Validierung des Modells ermöglicht darüber hinaus die Minimierung von Fehlern. Beim Altsystem werden im Gegensatz dazu Fehler meist nur zufällig entdeckt. Die modellgetriebene Softwareentwicklung scheint daher zur Umsetzung eines Systems zur Erstellung von virtuellen Ausstellungen geeignet zu sein.

Die Verwendung von GMF zur Umsetzung des Systems erscheint ebenfalls sinnvoll, trotz der in GMF enthaltenen Bugs. Zunächst unterstützt GMF die Entwicklung einer Rich-Client-Anwendung. Diese stellt dabei nur die Funktionen zur Verfügung, die für die Erstellung einer virtuellen Ausstellung erforderlich sind. Unter Verwendung von GMF konnten darüber hinaus alle Funktionsanforderungen erfüllt werden. Zusätzlich wurden außerdem zwei Wunschkriterien, das Anpassen der Größe von Bildern und die Überprüfung des Modells auf strukturelle Vollständigkeit, umgesetzt. Insbesondere die Überprüfung des Modells auf strukturelle Vollständigkeit wurde durch den von GMF generierten Validierungsmechanismus unterstützt. Des Weiteren ist die Veröffentlichung der nächsten Version von GMF für September diesen Jahres geplant.[1] Das Framework befindet sich also in der Entwicklung, und einige Bugs werden möglicherweise mit der nächsten Version behoben werden.

Die Entscheidung, den Codegenerator für den Prototyp speziell zu konzipieren und zu entwickeln, führte dazu, dass der Generator zwar auf die speziellen Anforderungen des Prototyps zugeschnitten ist. Es musste jedoch auch jede Funktion explizit entwickelt und getestet werden. Darüber hinaus hat die Entwicklung des Prototyps gezeigt, dass der Codegenerator mehr Funktionalität als angenommen, zum Beispiel eine M2M-Transformation, umsetzen muss. Im Hinblick auf die weitere Entwicklung des Codegenerators wird daher erwogen, oAW für die Codegenerierung zu verwenden. Bei der Verwendung dieses Frameworks müssten dabei die für die Erweiterung des Codegenerators benötigten Funktionen nicht explizit entwickelt werden.

[1] Siehe hierzu http://www.eclipse.org/projects/project_summary.php?projectid=modeling.gmf.

Kapitel 10.

Ausblick

Der im Rahmen dieser Diplomarbeit konzipierte und umgesetzte Prototyp dient als Grundlage für die weitere Entwicklung einer Ausstellungssoftware am MPIWG. Die Entwicklerin wird zu diesem Zweck ein weiteres halbes Jahr am MPIWG angestellt sein. Die weitere Entwicklung umfasst dabei die Umsetzung der noch ausstehenden Wunschkriterien und die Erweiterung des Metamodells. Eine Erweiterung des Metamodells ist notwendig, da der Prototyp noch nicht alle Funktionen des Altsystems zur Verfügung stellt. Die Erstellung eines Übersichtsplans für eine virtuelle Ausstellung beispielsweise wird noch nicht unterstützt. Darüber hinaus sind die folgenden Probleme des Prototyps zu beheben:

- Die Benutzeroberfläche darf bei der Ausführung der Zoom-Funktion nach der Validierung eines Modells nicht einfrieren.
- Der Prototyp ist trotz der Verwendung von Java nur auf dem Betriebssystem OS X lauffähig. Der Grund hierfür konnte dabei noch nicht eindeutig festgestellt werden. Die Ausstellungssoftware sollte jedoch auch auf anderen Betriebssystemen ausführbar sein, da Wissenschaftler aus anderen Institutionen oftmals nicht mit OS X sondern beispielsweise mit Windows arbeiten. Der Versuch, den Prototyp unter Verwendung des Eclipse-Plugins *Delta Pack* für Windows zu portieren, scheiterte. Das Plugin Delta Pack ermöglicht hierbei das Erstellen von Eclipse RCP-Anwendungen für unterschiedliche Plattformen.[1]

Neben den zu behebenden Problemen gibt es außerdem einige Eigenschaften des Prototyps, die bei der weiteren Entwicklung überdacht werden sollten. Hierfür sollten Befragungen und Diskussionen mit den Fachexperten durchgeführt werden. Folgende Punkte werden in Bezug auf eine Überarbeitung zur Diskussion gestellt:

- Der Prototyp speichert alle Modellelemente in einer Modell-Datei. Bei sehr vielen Modellelementen wird diese Datei entsprechend groß. Die Möglichkeit, den Inhalt der Modell-Datei auf mehrere Modell-Dateien zu verteilen, sollte diskutiert werden.

[1] Das Plugin Delta Pack muss zusätzlich in Eclipse installiert werden. Es ist zu finden unter http://www.software-mirror.com/eclipse/eclipse/downloads/drops/R-3.2.1-200609210945/.

- Unterschiedliche Editoren, die Teile desselben Modells abbilden, sollten dieselbe Editing Domain nutzen.
- Die Darstellung des Inhalts eines Exhibition Modules sollte überarbeitet werden. Die Darstellung, die für den Prototyp umgesetzt wurde, ist bei einer hohen Anzahl von Slide, Branching Points und Sequences sehr unübersichtlich. Möglicherweise wäre eine Listenansicht oder Ähnliches sinnvoller.

Darüber hinaus sollten weiterhin Anwendertests durchgeführt werden. Diese Tests würden der weiteren Analyse der Ausstellungssoftware dienen. Insbesondere das Verhalten der Software bei der Erstellung von umfangreicheren virtuellen Ausstellungen, als im Rahmen dieser Diplomarbeit erzeugt wurden, sollte untersucht werden. Die Anwendertests könnten bei entsprechender Weiterentwicklung der Ausstellungssoftware dabei unter anderem als *Betatests* konzipiert werden. Im Rahmen der für diese Diplomarbeit durchgeführten Anwendertests wurde von einigen Testpersonen bereits großes Interesse diesbezüglich geäußert.

«Die Zeit blühte auf, die Materie schrumpfte weg. Die größte existierende Primzahl verkrümelte sich geräuschlos in einer Ecke und versteckte sich für immer.»

(Douglas Adams, aus: Per Anhalter durch die Galaxis.)

Anhang A.

Ergänzungen zur modellgetriebenen Softwareentwicklung

A.1. Ergänzungen zur Codegenerierung mittels einer Programmiersprache

Listing A-1 zeigt beispielhaft die Implementierung einer Klasse zur Generierung von Java-Klassen.

```
1  public class JavaClass {
2          public String packageName;
3          public String name;
4          public List<JavaMember> members;
5
6          @Override
7          public String toString()
8          {
9              String buff = "package " + packageName + ";\n" +
10                 "\n" +
11                 "public class " + name + " {\n";
12             for (Member m : members)
13                 buff += m;
14             buff += "}\n";
15             return buff;
16         }
17 }
```

Listing A-1 Beispielklasse eines Generators zur Erstellung von Java-Klassen, Quelle: ([Sta07]), S. 150

A.2. Elemente der MOF

Abbildung A-1 zeigt einen vereinfachten Ausschnitt der MOF.

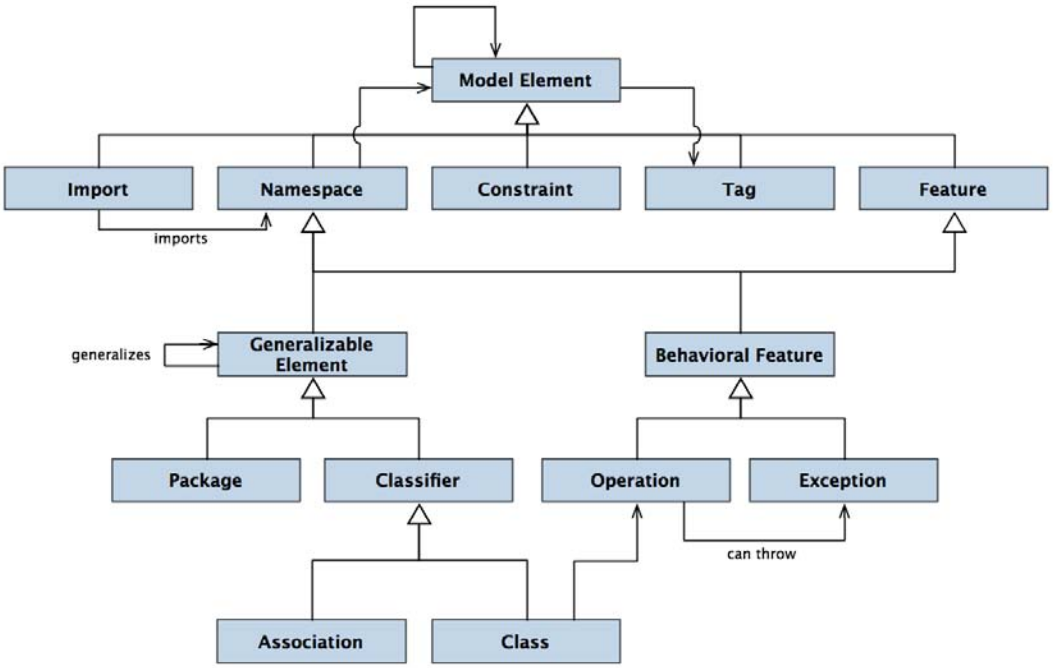


Abb. A-1. Ausschnitt der MOF, Quelle: ([Sta07], S. 65)

A.3. Elemente der EMOF

Abbildung A-2 zeigt einen vereinfachte Darstellung der Elemente der EMOF.

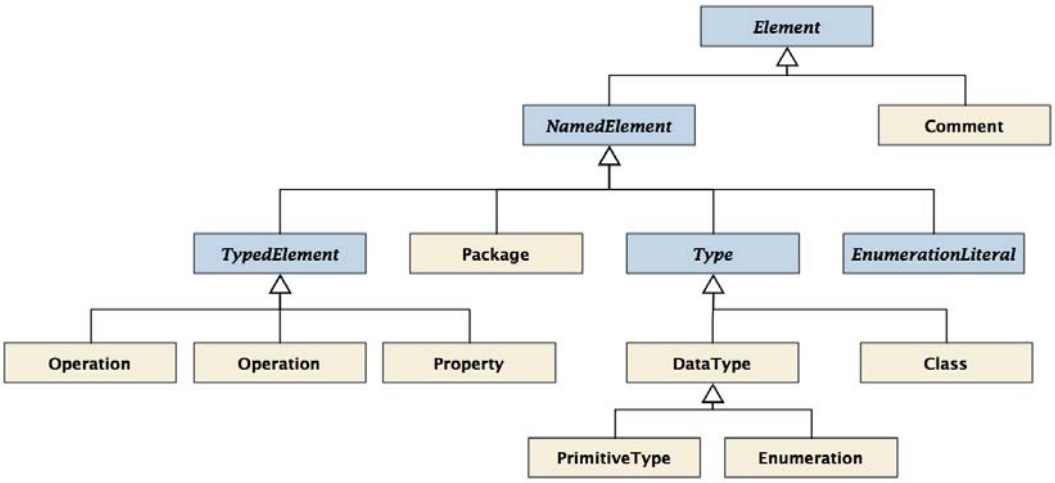


Abb. A-2. Elemente der EMOF, Quelle: angelehnt an die Klassen-Diagramme in ([OMG06])

A.4. Ergänzungen zu JET

A.4.1. Codegenerierungsansatz von JET

Listing A-2 zeigt ein JET-Template.

```
Hello, <%=argument%>!
<% stringBuffer.append("Hello again!"); %>
```

Listing A-2 Einfaches JET-Template, Quelle: ([Pop04], Abschnitt: Under the Hood)

Listing A-3 zeigt die Java-Klasse, die für das zuvor gezeigte Template durch JET generiert wird.

```
package hello;

public class TranslationDemoTemplate
{
   protected final String NL = System.getProperties().getProperty("line.
       separator");
   protected final String TEXT_1 = "Hello, ";
   protected final String TEXT_2 = "!";

   public String generate(Object argument)
   {
     StringBuffer stringBuffer = new StringBuffer();
     stringBuffer.append(TEXT_1);
     stringBuffer.append(argument);
     stringBuffer.append(TEXT_2);

     stringBuffer.append("Hello again!");
     return stringBuffer.toString();
   }
}
```

Listing A-3 Für das JET-Template generierte Java-Klasse, Quelle: ([Pop04], Abschnitt: Under the Hood)

A.4.2. JET Templates

Listing A-4 zeigt ein Modell im XML-Format. Aus diesem Modell kann mittels des JET-Templates in Listing A-5 eine Java-Klasse erstellt werden. Das JET-Template in Listing A-6 erstellt aus dem Modell eine Ruby-Klasse.

```
<class name="HelloClass">
   <phrase>Hello,World!</phrase>
</class>
```

Listing A-4 Beispiel eines XML-Modells, Quelle: ([Gov07])

```
public class <c:get select="/class/@name" /> {
   public static void main(String[] args) {
      System.out.println("<c:get select="/class/phrase" />");
   }
}
```

Listing A-5 Beispiel eines JET-Templates zur Erzeugung einer Java-Klasse, Quelle: ([Gov07])

```
class <c:get select="/class/@name" />
  def sayPhrase
    puts "<c:get select="/class/phrase" />"
  end
end
```

Listing A-6 Beispiel eines JET-Templates zur Erzeugung einer Ruby-Klasse, Quelle: ([Gov07])

Anhang B.

Ergänzungen zum Altsystem

B.1. Beispiele virtueller Ausstellungen

B.2. Beispiele für Scenes

Nachfolgend einige Screenshots von Scenes virtueller Ausstellungen.

B.2.1. Virtuelle Einsteinausstellung

Nachfolgend ein Screenshot der virtuellen Einsteinausstellung. Die virtuelle Einsteinausstellung ist die Abbildung eines realen Raums. Abbildung B-2 zeigt die Eingangshalle des Kronprinzenpalais in Berlin, in dem die Ausstellung "Albert Einstein - Ingenieur des Universums" ausgerichtet wurde.

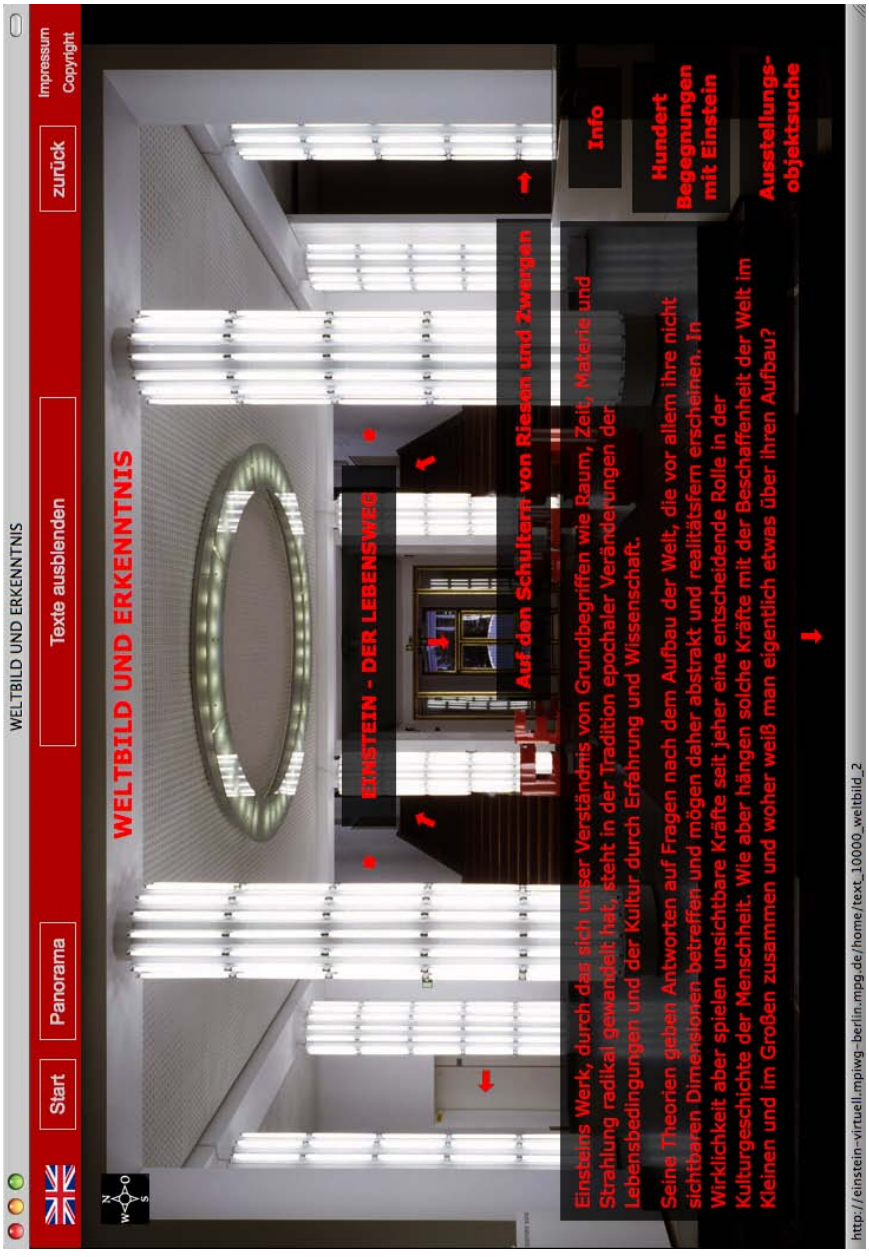


Abb. B-1. Virtuelle Einsteinausstellung

B.2.2. Virtuelle Kanarische Inseln

Der nachfolgende Screenshot zeigt ein Beispiel für eine virtuelle Ausstellung, die einen abstrakten Raum abbildet. Über Pfeile ist die Navigation zwischen Landkarten der Kanarischen Inseln möglich. Diese virtuelle Ausstellung wurde zu Demonstrationszwecken für eine Schulung entwickelt und ist nicht mehr im Internet verfügbar.

Abb. B-2. Virtuelle Kanarische Inseln

B.3. Ergänzende Erläuterungen zur Nomenklatur

Nachfolgend einige erläuternde Screenshots zur Nomenklatur.

B.3.1. Branching Point

Abbildung B-3 zeigt einen Branching Point mit vier Branching Point Choices.

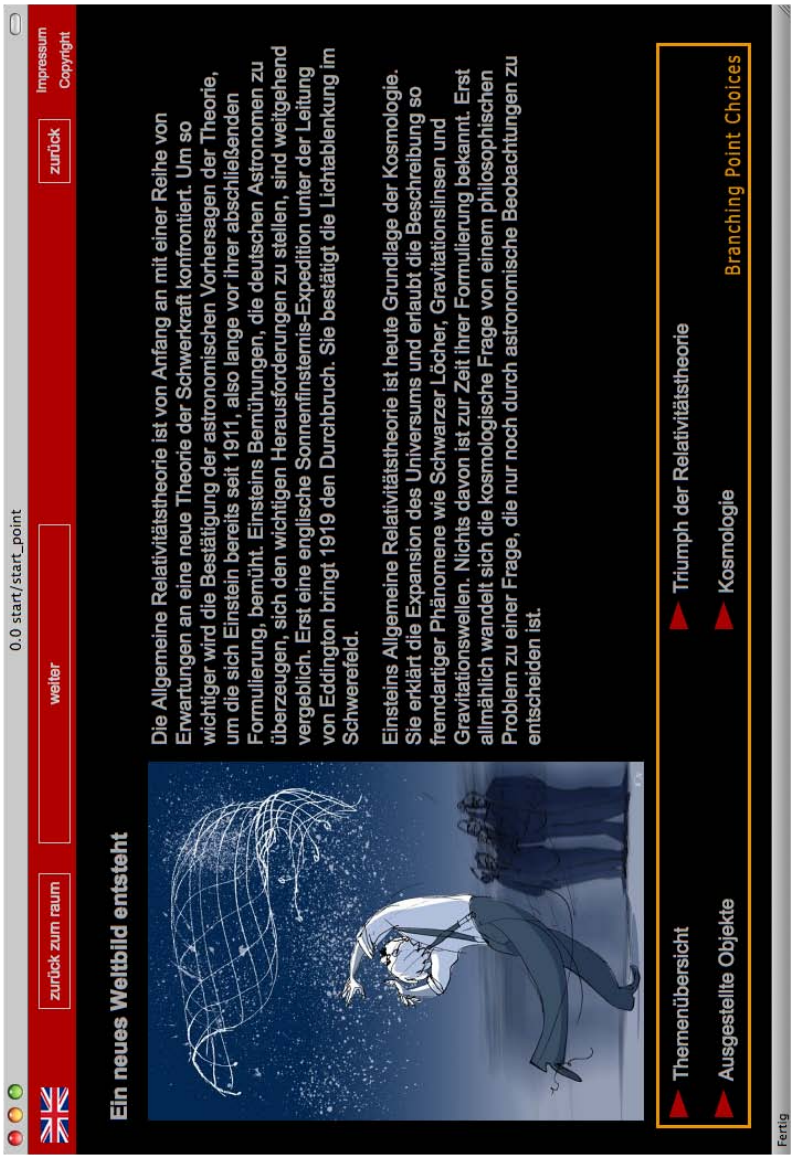


Abb. B-3. Branching Point

B.3.2. Scene

Abbildung B-4 zeigt eine Scene mit einem Exhibition Module Link und zwei Scene Links. Die Scene Links haben jeweils zwei Darstellungsformen. Sie sind durch ein Textfeld mit dem Titel des Links repräsentiert und durch einen Pfeil.

Abb. B-4. Scene mit zwei Scene Links und einem Exhibition Module Link

B.4. Ergänzungen zu den Implementierungsdetails

Nachfolgend einige Ergänzungen und Screenshots zu den Implementierungsdetails des Altsystems.

B.4.1. Übersicht über alle Exhibition Modules

Abbildung B-5 zeigt einen Screenshot des Webinterfaces für die Verwaltung der Exhibition Modules. Der Screenshot zeigt die Übersicht über alle existierenden Exhibition Modules.

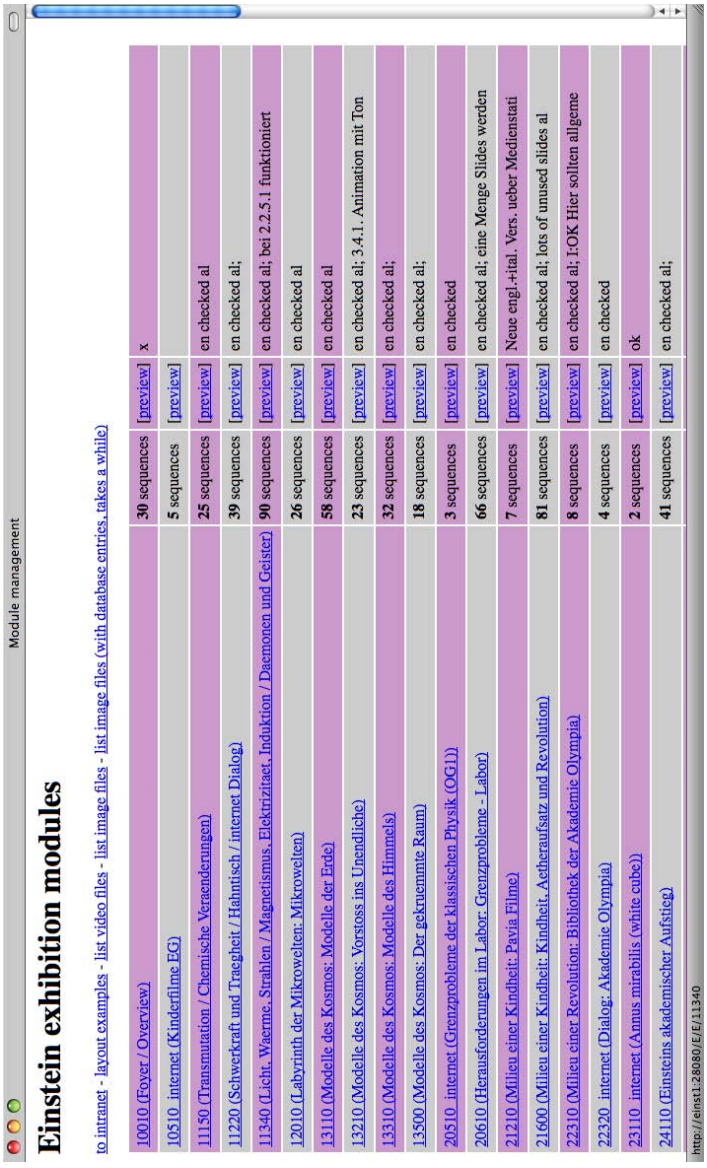


Abb. B-5. Verwaltung der Exhibition Modules

B.4.2. Sequence

Abbildung B-6 zeigt einen Screenshot des Web-Interfaces für die Erstellung einer Sequence. In der linken oberen Ecke befindet sich ein Klappmenü mit den Aktionen, die der Benutzer durchführen kann, beispielsweise das Hinzufügen eines neuen Branching Points in die Sequence. Im unteren Teil des Fensters ist der erste Slide der Sequence zu sehen.

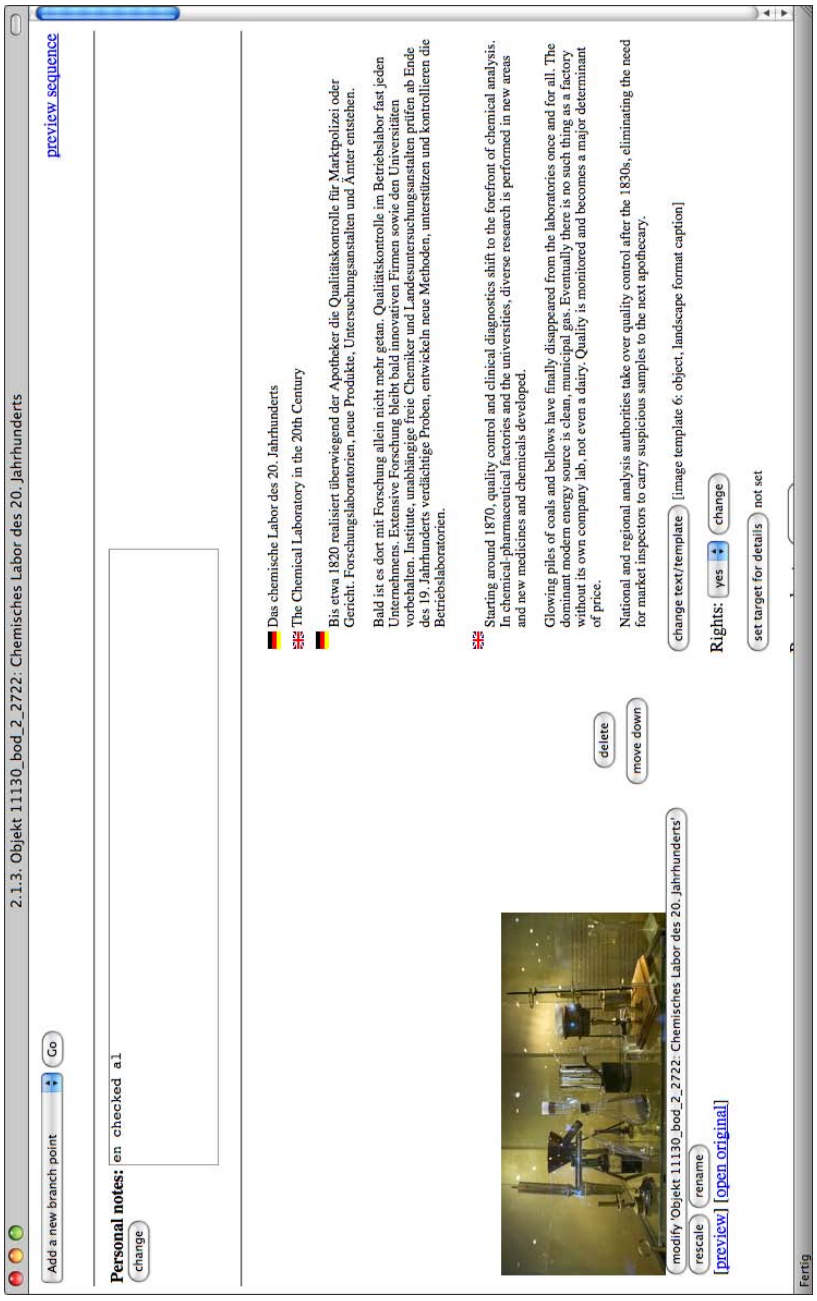


Abb. B-6. Erstellung einer Sequence

B.4.3. Hinzufügen eines Bildes

Abbildung B-7 zeigt einen Screenshot des Web-Interfaces für das Hinzufügen eines Bildes zu einem Slide oder Branching Point. In das gelbe Feld kann per "Drag and Drop" ein Bild aus einer ImageCollection eingefügt werden.

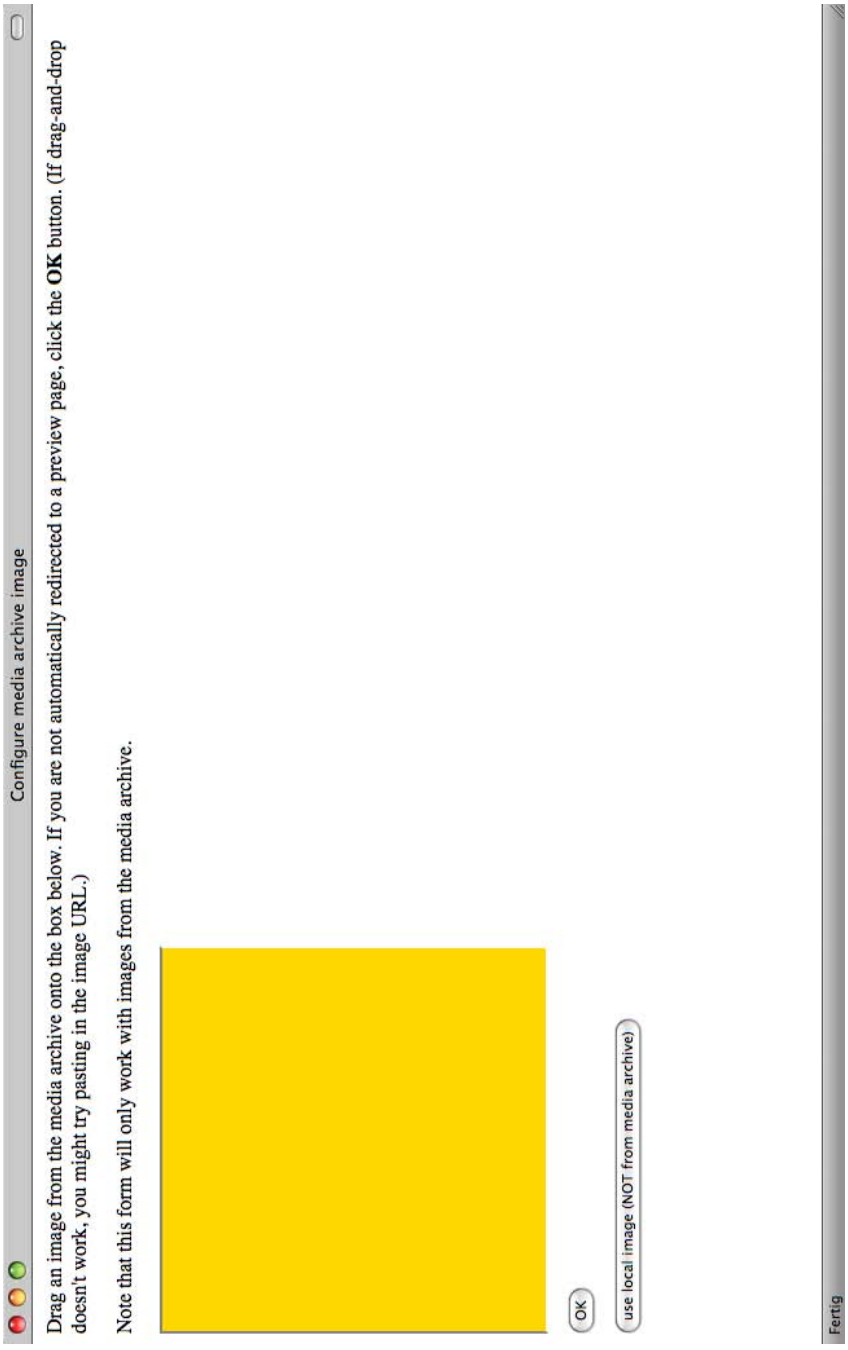


Abb. B-7. Hinzufügen eines Bildes zu einem Slide oder Branching Point

B.4.4. Template für die Anzeige von Scenes

Listing B-1 zeigt ein ZPT-Template zur Anzeige der Scenes in der virtuellen Einsteinausstellung.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.
    w3.org/TR/html4/loose.dtd">
<html tal:attributes="lang here/getLanguage | string:de" metal:define-macro
    ="page">
<head metal:define-slot="head">
  <tal:block tal:define="global weiter python:here.REQUEST.cookies.get('
      _next','yes')"/>
  <link href="virtuelleAus.css" rel="stylesheet" type="text/css">
  <link href="mapstyle_css" rel="stylesheet" type="text/css">
  <meta http-equiv="content-type" content="text/html; charset=latin-1">
  <title metal:define-slot="title"><tal:block tal:replace="here/title"/></
      title>

  <tal:block metal:define-slot="script">
  <tal:block tal:condition="python:weiter=='no'"><tal:block tal:define="
      dummy python:here.REQUEST.set('map_force_types','arrow')"/></tal:
      block>
  <tal:block tal:condition="python:weiter=='yes'"><tal:block tal:define="
      dummy python:here.REQUEST.set('map_force_types','')"/></tal:block>
    <script tal:replace="structure here/createMapHead"/>
  </tal:block>

</head>
<body tal:condition="not:python:here.absolute_url()+'/index_html'==here.
    REQUEST['URL']"><tal:block tal:define="dummy python:here.REQUEST.
    response.redirect(here.absolute_url()+'?'+here.REQUEST['QUERY_STRING'])
    "/>
</body>
<body tal:condition="python:here.absolute_url()+'/index_html'==here.REQUEST
    ['URL']" metal:define-slot="body" onload="map_init()">
  <!-- navigation arrowa -->
    <tal:block tal:define="left python:here.getVD('left'); right python:
        here.getVD('right'); out python:here.getVD('out'); main python:here
        .getVD('main');">
    <div tal:condition="out" style="position:absolute;top:490pt;left:350pt"
        >
            <a class="image" tal:attributes="href out" class="image"><img
                src="arrows/down_2.gif" border="0"></a>
    </div>
    </tal:block>


<!-- content -->
<table class="content" padding=0 cellspacing=0 border=0 width="100%" height
    ="680px" style="margin-top: 0px;z-index:0;">
```

```
31  <tr><td height="39px">
32
33      <!-- navigation -->
34      <div id="nav_bottom" style="z-index: 50;">
35      <table width="100%" id="nav" background="#B00000" border=0
            cellspacing=0 cellpadding=0>
36      <tr>
37         <td width="40px">
38            <!-- language switcher with flag button -->
39            <a tal:condition="python:here.getLanguage()=='de'" id="flag"
                 href="./switchLanguage">
40               <img id="flag" tal:attributes="src python:'images/flag_en.
                     gif'; alt python:context.getLanguage()" width="50"
                     height="25" style="float:left">
41            </a>
42            <a tal:condition="python:here.getLanguage()=='en'" id="flag"
                 href="./switchLanguage">
43               <img id="flag" tal:attributes="src python:'images/flag_de.
                     gif'; alt python:context.getLanguage()" width="50"
                     height="25" style="float:left">
44            </a>
45         </td>
46         <td width="70px">
47            <!-- home -->
48            <a href="home">Start</a>
49         </td>
50
51         <td width="50px">
52            <tal:block tal:define="panorama here/getPanorama">
53                  <a  target="_blank" tal:condition="panorama" tal:
                        attributes="href panorama">Panorama</a>
54            </tal:block>
55         </td>
56         <td> </td>
57         <td align="center">
58            <tal:block tal:condition="python:weiter=='no'">
59                     <a id="next" tal:condition="python:here.getLanguage()
                           =='de'" tal:attributes="href python:here.
                           absolute_url()+'/setWeiter'">Ausstellungsinhalte</
                           a>
60                     <a id="next" tal:condition="python:here.getLanguage()
                           =='en'" tal:attributes="href python:here.
                           absolute_url()+'/setWeiter'">Content</a>
61            </tal:block>
62         </td>
63         <td align="center">
64            <tal:block tal:condition="not:python:weiter=='no'">
65                     <a id="next" tal:condition="python:here.getLanguage()
                           =='de'" tal:attributes="href python:here.
                           absolute_url()+'/clearDisplay'">Texte 
                           ausblenden</a>
```

```
66                      <a id="next" tal:condition="python:here.getLanguage()
                            =='en'" tal:attributes="href python:here.
                            absolute_url()+'/clearDisplay'">Hide Texts</a>
67              </tal:block>
68
69              <tal:block metal:define-slot="page_foot"/>
70          </td>
71
72          <td width="50px">
73              <a tal:condition="python:here.getLanguage()=='de'" style="float
                    :right" href="javascript:history.back()">zur&uuml;ck</a>
74              <a tal:condition="python:here.getLanguage()=='en'" style="float
                    :right" href="javascript:history.back()">back</a>
75          </td>
76          <td width="50px" align="center">
77              <table>
78                  <tr><td align="center"><a href="http://www.mpiwg-berlin.mpg.
                        de/de/impressum.html" target="_blank" id="rights">
                        Impressum</a></td></tr>
79                  <tr><td align="center"><a href="copyrights" target="_blank"
                        id="rights">Copyright</a></td></tr>
80              </table>
81          </td>
82
83      </tr>
84      </table>
85      </div>
86
87      <!-- ende navigation -->
88  </td></tr>
89
90  <tr bgcolor="black">
91
92    <td valign="top" height="640px">
93     <div tal:replace="structure python:here.createMapImg()"/>
94    </td>
95  </tr>
96
97 </table>
98 <!-- title -->
99
100     <div  class="maptext" tal:condition="python:weiter=='yes'" style="
            position:absolute;top:65px;left:0px;width:1024px;"><h1 tal:
            content="structure python:here.getTitle()">HALLO</h1></div>
101
102 <!-- link block -->
103     <div style="display: none">
104     <tal:block tal:repeat="ob python:here.getAllMapAreas(mapColTypes=['
            ECHO_collection','ECHO_resource','ECHO_link'])">
105     <a tal:replace="structure python:here.createMapLink(ob,target='_top')
            "/><br/>
```

```
106         </tal:block>
107         <tal:block tal:repeat="ob python:here.getAllMapAreas(mapColTypes=['
                ECHO_externalLink'])">
108         <a tal:replace="structure python:here.createMapLink(ob,target='_blank
                ')"/><br/>
109         </tal:block>
110         </div>
111 <!-- end of link block -->
112
113 <!-- auxiliary image block -->
114
115       <tal:block tal:repeat="ob python:here.getAllMapAreas(mapColTypes=['
              ECHO_collection','ECHO_resource','ECHO_link'])">
116         <a tal:replace="structure python:here.decode(here.createMapAux(ob,
              target='_top',circlesrc='http://nausikaa2.rz-berlin.mpg.de/
              digitallibrary/servlet/Scaler/?dw=15&fn=/permanent/icons/kreis',
               backLink=here.REQUEST.get('backLink',None)))" />
117       </tal:block>
118         <tal:block tal:repeat="ob python:here.getAllMapAreas(mapColTypes=['
              ECHO_externalLink'])">
119         <a tal:replace="structure python:here.decode(here.createMapAux(ob,
              circlesrc='http://nausikaa2.rz-berlin.mpg.de/digitallibrary/
              servlet/Scaler/?dw=15&fn=/permanent/icons/kreis',target='_blank')
              )"/>
120       </tal:block>
121 <!-- end of auxiliary image block -->
122 </body>
123 </html>
```

Listing B-1 ZPT-Template zur Anzeige von Scenes

B.4.5. ImageCollection

Abbildung B-8 zeigt einen Screenshot einer ImageCollection. Im linken Teil des Browserfensters wird eine Übersicht der durch die ImageCollection verwalteten Bilder angezeigt. Im rechten Teil des Browserfensters werden ein ausgewähltes Bild und dessen Metadaten angezeigt. Das Bild kann über die darüberliegende Menüleiste vergrößert, verkleinert, gedreht werden etc.

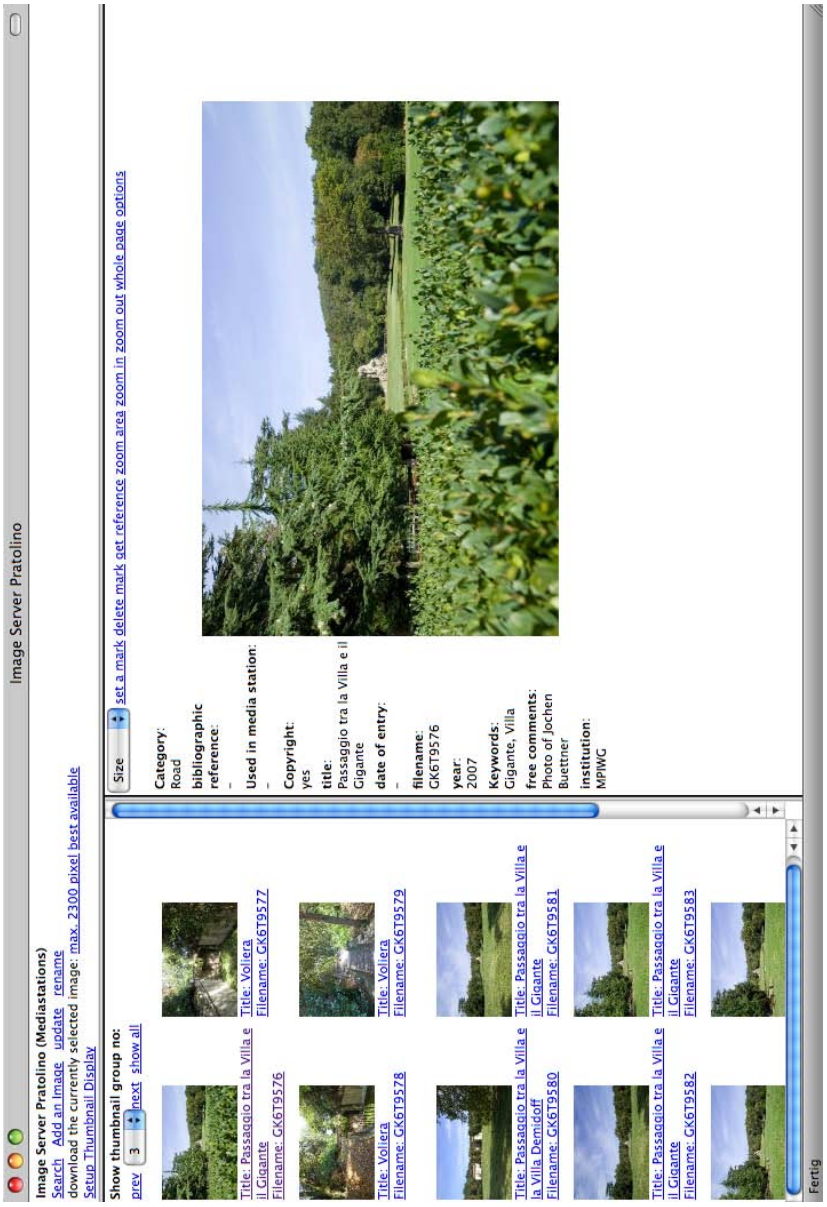


Abb. B-8. ImageCollection

Anhang C.

Ergänzungen zur Anforderungsanalyse

Im Folgenden Ergänzungen zur Anforderungsanalyse.

C.1. Funktionsanforderungen

Nachfolgend eine detaillierte Aufstellung der Funktionsanforderungen, die an das System gestellt werden.

C.1.1. Virtuelle Ausstellung

Die Software ermöglicht das Erstellen und Ändern einer virtuellen Ausstellung.

/FA0100/ Erstellen einer virtuellen Ausstellung

Es kann eine virtuelle Ausstellung erstellt werden. Eine virtuelle Ausstellung besteht aus Scenes und Exhibition Modules. Die Scenes und Exhibition Modules können mittels der Software verwaltet werden. Eine virtuelle Ausstellung kann mit Hilfe von HTML-Seiten und JavaScript im Internet präsentiert werden.

/FA0101/ Ändern einer virtuellen Ausstellung

Eine virtuelle Ausstellung kann geändert werden, indem Scene oder Exhibition Modules hinzugefügt, geändert oder entfernt werden.

/FA0102/ Löschen einer virtuellen Ausstellung

Eine virtuelle Ausstellung kann gelöscht werden. Dabei werden alle enthaltenden Scenes und Exhibition Modules ebenfalls gelöscht.

C.1.2. Verwaltung von Scenes

Die Software ermöglicht das Erstellen, Ändern und Löschen von Scenes (siehe Abbildung C-3).

/FA0200/ Erstellen einer Scene

Es kann eine Scene erstellt werden. Der Scene wird ein Titel und eine Beschreibung gegeben und ein Hintergrundbild zugewiesen.

/FA0201/ Ändern einer Scene

Bei einer Scene kann der Titel, die Beschreibung und das Hintergrundbild kann geändert werden.

/FA0202/ Scene Links

Von einer Scene kann auf andere Scenes mittels Scene Links verwiesen werden. Ein Scene Link ist dabei unidirektional. Für ein Scene Link wird eine Ausgangs-Scene und eine End-Scene festgelegt (siehe Abbildung 6-9). Des Weiteren wird ein Titel für einen Scene Link angegeben.

/FA0203/ Löschen einer Scene

Eine Scene kann gelöscht werden. Das Löschen einer Scene führt dazu, dass alle Scene Links, bei denen die Scene Ausgangs- oder End-Scene ist, ebenfalls gelöscht werden.

C.1.3. Verwaltung von Exhibition Modules

Die Software ermöglicht das Erstellen, Ändern und Löschen von Exhibition Modules (siehe Abbildung 6-10). Im Folgenden werden die dafür notwendigen funktionalen Anforderungen beschrieben.

/FA0300/ Erstellen von Exhibition Modules

Es kann ein Exhibition Module erstellt werden. Dafür wird ein Titel angegeben. Einem Exhibition Module können Slides, Branching Points und Sequences zugefügt werden.

/FA0301/ Ändern von Exhibition Modules

Der Titel eines Exhibition Modules kann geändert werden. Es können weitere Slide, Branching Points und Sequences hinzugefügt oder bestehende gelöscht werden.

/FA0302/ Exhibition Module Links

Auf ein Exhibition Module kann von einer Scene mittels eines Exhibition Module Links verwiesen werden. Der Verweis ist bidirektional (siehe Abbildung 6-11). Es wird ein Titel für den Exhibition Module Link angegeben.

/FA0303/ Löschen von Exhibition Modules

Exhibition Modules können gelöscht werden. Dabei werden alle hinzugefügten Slides, Branching Points und Sequences ebenfalls gelöscht. Wird auf das Exhibition Module von einer oder mehrerer Scenes verwiesen, wird bzw. werden die Exhibition Module Links ebenfalls gelöscht.

/FA0304/ Festlegen der Start-Sequence

Für ein Exhibition Module kann eine Start-Sequence festgelegt werden. Durch die Start-Sequence wird der Slide eines Exhibition Modules der beim Aufruf des Exhibition Modules angezeigt wird festgelegt. Die Start-Sequence kann jederzeit geändert werden.

/FA0305/ Erstellen von Slides

Es können Slides erstellt werden. Einem Slide wird ein Titel, ein Untertitel, beliebig viele Texte, beliebig viele Bilder und eine Fußzeile zugewiesen.

/FA0306/ Ändern von Slides

Der Titel, der Untertitel, die Texte, die Bilder und die Fußzeile eines Slides können geändert werden. Es können weitere Texte und Bilder hinzugefügt werden. Bereits hinzugefügte Texte und Bilder können entfernt werden.

/FA0307/ Löschen von Slides

Slides können gelöscht werden. Dabei werden der Titel, Untertitel, die Texte, die Bilder und die Fußzeile ebenfalls gelöscht. Wird auf einen Slide von einer Sequence verwiesen, wird der Verweis gelöscht.

/FA0308/ Erstellen von Branching Points

Es können Branching Points erstellt werden. Dafür wird dem Branching Point ein Titel, ein Untertitel, beliebig viele Texte, beliebig viele Bilder und eine Fußzeile zugewiesen. Des Weiteren können dem Branching Point Branching Point Choices hinzugefügt werden. Eine Branching Point Choice hat einen Titel und verweist auf eine Sequence.

/FA0309/ Ändern von Branching Points

Der Titel, der Untertitel, die Texte, die Bilder und die Fußzeile eines Branching Points können geändert werden. Es können weitere Texte und Bilder hinzugefügt werden. Bereits hinzugefügte Texte und Bilder können entfernt werden. Des Weiteren können der Titel und der Verweis auf eine Sequence einer Branching Point Choice geändert werden. Es können neue Branching Point Choices hinzugefügt werden. Bereits hinzugefügte Branching Point Choices können entfernt werden.

/FA0310/ Löschen von Branching Points

Branching Points können gelöscht werden. Dabei werden der Titel, Untertitel, die Texte, die Bilder, die Fußzeile und die Branching Point Choices ebenfalls gelöscht. Wird auf einen Branching Point von einer Sequence verwiesen, wird der Verweis gelöscht.

/FA0311/ Erstellen von Sequences

Es können Sequences erstellt werden. Dafür wird ein Titel angegeben. Eine Sequence verweist auf Slides und Branching Points. Die Verweise können angegeben werden. Des Weiteren kann die Reihenfolge der Verweise festgelegt werden.

/FA0312/ Ändern von Sequences

Der Titel einer Sequence kann geändert werden. Des Weiteren können neue Verweise zu Slides und Branching Points hinzugefügt und bestehende entfernt werden. Die Reihenfolge der Verweise kann geändert werden.

/FA0313/ Löschen von Sequences

Sequences können gelöscht werden. Dabei werden der Titel und die Verweise zu Slides und Branching Points gelöscht.

C.2. Wunschkriterien

Abhängig von dem Fortschritt bei der Entwicklung der Software können die folgenden Kriterien zusätzlich umgesetzt werden.

- **Manuelle Anpassung der Bildgröße**
  Die Bildgröße der Bilder, die für eine virtuelle Ausstellung verwendet werden, kann mittels der Software geändert werden.

- **Automatische Überprüfung auf Vollständigkeit**
  Die Software ermöglicht die automatische Überprüfung einer virtuellen Ausstellung auf strukturelle Vollständigkeit. Der Anwender wird auf darauf hingewiesen, wenn beispielsweise für eine Scene kein Hintergrundbild angegeben ist oder ein Slide in keiner Sequence verwendet wird.

- **Verwaltung der Templates**
  Die Templates, die für die Slides und Branching Points verwendet werden, können über die Software verwaltet werden. Es können Templates hinzugefügt, gelöscht oder geändert werden. Für den Austausch von Templates zwischen unterschiedlichen Installationen der Software können Templates importiert und exportiert werden.

- **Erzeugung einer Ausstellungsdokumentation**
  Zusätzlich zu der erzeugten Website soll die Erzeugung einer PDF-Datei mit den Inhalten der Website ermöglicht werden.

- **Verwaltung der Bilder**
  Die in der virtuellen Ausstellung verwendeten Bilder sollen über die Software verwaltet werden können. Bilder können mit Metadaten versehen werden. Die Metadaten der Bilder können durchsucht werden. Zu einem Bild wird angezeigt, in welchem Slide, Branching Point oder in welcher Scene das Bild verwendet wird.

- **Verwendung einer ImageCollection**
  Es soll ermöglicht werden in der virtuellen Ausstellung verwendete Bilder über eine ImageCollection zu verwalten.

- **Audio- und Videomaterial**
  Neben Bildern können in einem Slide auch Audio- oder Video-Dateien eingebunden werden.

- **Icons zur Navigation**
  Die Links innerhalb einer Scene zu anderen Scenes können durch Icons dargestellt werden.

C.3. Abgrenzungskriterien

Die folgenden Kriterien werden von der Software nicht erfüllt.

- **Erstellung von Animationen**
  Es können keine Animationen mittels der Software erstellt werden.

- **Erstellung von Videos und Bildern**
  Die Software ermöglicht nicht die Erstellung Videos und Bildern. Videos können durch die Software nicht bearbeitet werden.

- **Erweiterte Bildbearbeitung**
  Bilder können durch die Software nur in der Größe verändert werden. Darüber hinaus gehende Bildbearbeitungsfunktionen werden durch die Software nicht umgesetzt.

C.4. Produkteinsatz

C.4.1. Anwendungsbereich

Die Software wird am MPIWG und dessen Partnerinstitutionen zur Erstellung von virtuellen Ausstellungen eingesetzt.

C.4.2. Zielgruppe

Die Anwender der zu entwickelnden Software sind Wissenschaftler. Diese Wissenschaftler arbeiten am MPIWG und bei Partnerinstitutionen des MPIWGs. Sie stammen aus unterschiedlichen Ländern und haben unterschiedliche Forschungsgebiete. Die Anwender arbeiten am Computer. Sie haben jedoch keine tiefergehenden technischen Kenntnisse. Für die Bedienung der Software wird die Beherrschung der englischen Sprache vorausgesetzt.

C.4.3. Technische Produktumgebung

Für den Betrieb der Software wird ein Rechner mit 1 GHz Prozessorleistung, 1 GB RAM, 60 MB bis 100 MB freier Festplattenspeicher und dem Betriebssystem OS X von Apple vorausgesetzt.

C.5. Use-Case-Diagramme

Im Folgenden befinden sich weitere Use-Case-Diagramme. Die Diagramme ergänzen die in Kapitel 6 aufgeführten funktionalen Anforderungen.

/FA0314/ Erstellen einer Scene

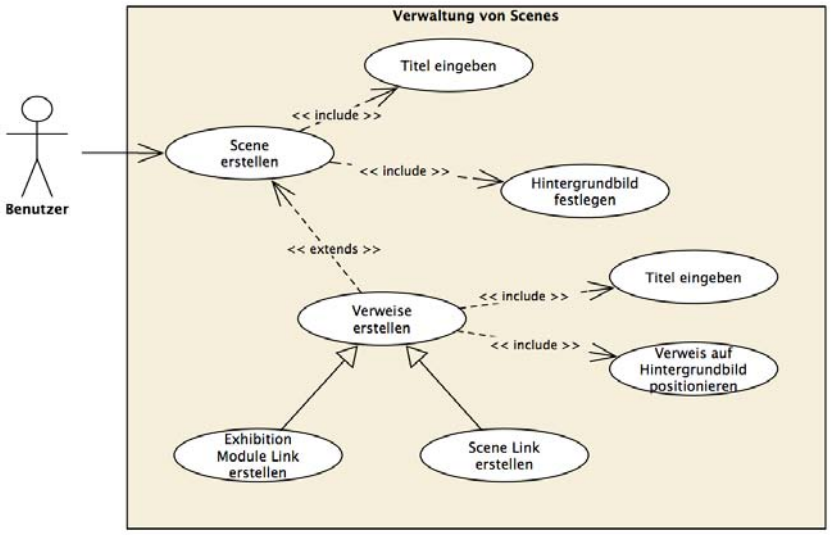


Abb. C-1. Use-Case Scenes erstellen

/FA0314/ Ändern einer Scene

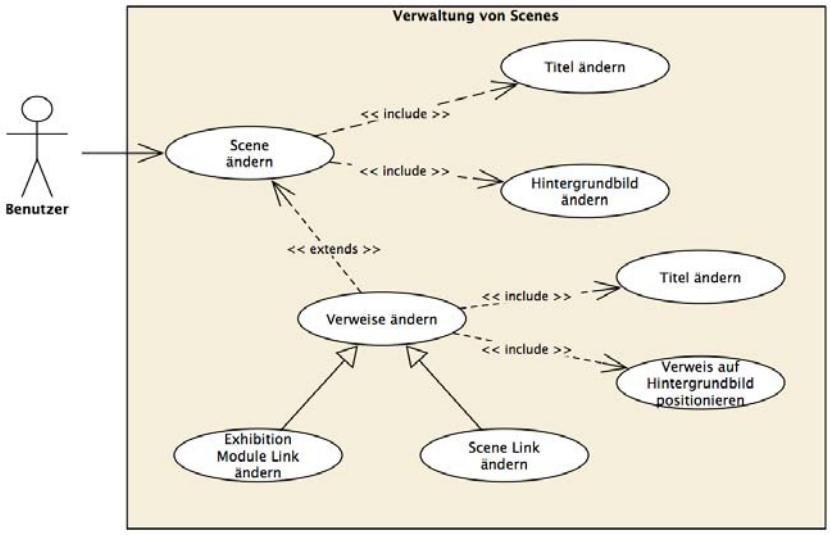


Abb. C-2. Use-Case Scenes ändern

/FA0314/ Erstellen von Exhibition Modules

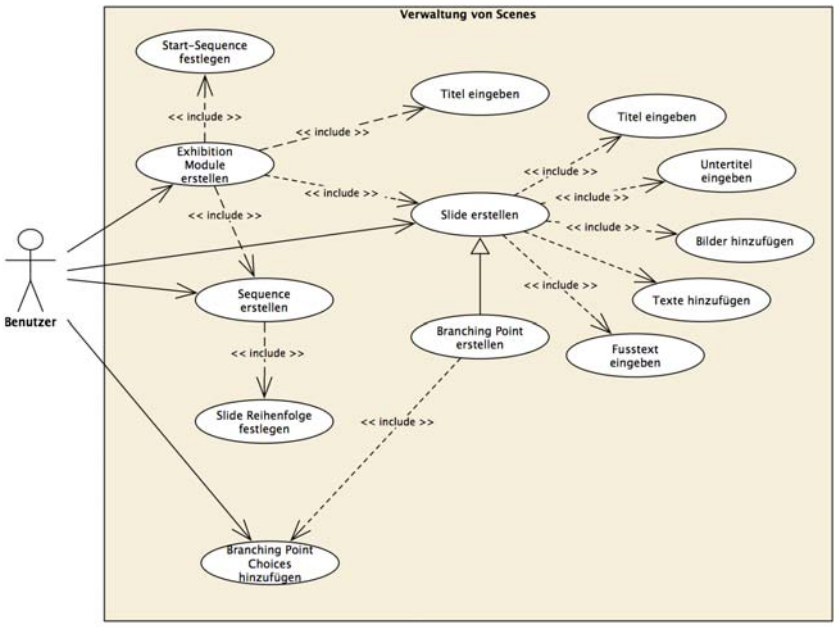


Abb. C-3. Use-Case Exhibition Module erstellen

Anhang D.

Ergänzungen zur Konzeption des Systems

D.1. Domänenmodell

Abbildung D-1 zeigt das Domänenmodell des Prototyps.

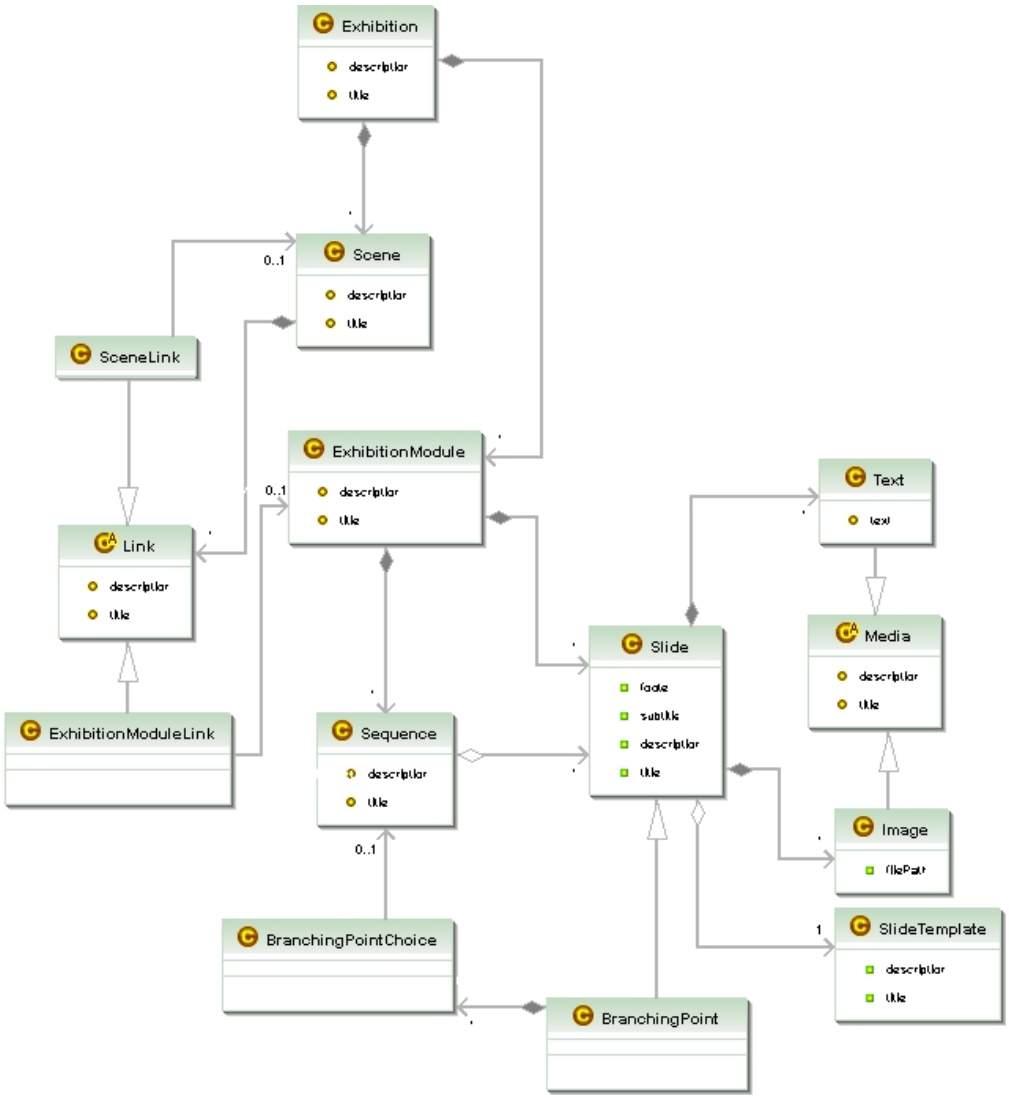


Abb. D-1. Domänenmodell des Prototyps

D.2. Ergänzungen zum Metamodell

D.2.1. Quelltext des Metamodells

Listing D-1 zeigt das vollständige Metamodell des Prototyps als Ecore-Datei.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
        XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="exhibition"
    nsURI="http:///org/jat/exhibition.ecore" nsPrefix="org.jat.exhibition">
  <eClassifiers xsi:type="ecore:EClass" name="Audio" eSuperTypes="#//Media"
      >
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="audioFile" eType
        ="#//File"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="BranchingPoint" eSuperTypes="
      #//Slide">
    <eStructuralFeatures xsi:type="ecore:EReference" name="choices"
        upperBound="-1"
        eType="#//BranchingPointChoice" containment="true" resolveProxies="
            false"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
        numberOfColumnsOfChoices"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt
            "/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="BranchingPointChoice">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
        ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="sequence" eType=
        "#//Sequence"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Category">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="categoryTitle"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
        EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="icon" eType="#//
        Image"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Exhibition">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
        ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
        EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="exhibitionMap"
        eType="#//ExhibitionMap"
        containment="true" resolveProxies="false"/>
```

```
28      <eStructuralFeatures xsi:type="ecore:EReference" name="scenes"
            upperBound="-1"
29          eType="#//Scene" containment="true" resolveProxies="false"/>
30      <eStructuralFeatures xsi:type="ecore:EReference" name="
            exhibitionModules" upperBound="-1"
31          eType="#//ExhibitionModule" containment="true" resolveProxies="
                false"/>
32      <eStructuralFeatures xsi:type="ecore:EReference" name="startScene"
            eType="#//Scene"/>
33    </eClassifiers>
34    <eClassifiers xsi:type="ecore:EClass" name="ExhibitionMap">
35      <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
36      <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
37      <eStructuralFeatures xsi:type="ecore:EReference" name="mapImage" eType=
            "#//Image"
38          containment="true" resolveProxies="false"/>
39      <eStructuralFeatures xsi:type="ecore:EReference" name="linksToScenes"
            upperBound="-1"
40          eType="#//SceneLink" containment="true" resolveProxies="false"/>
41      <eStructuralFeatures xsi:type="ecore:EReference" name="
            linksToExhibitionModules"
42          upperBound="-1" eType="#//ExhibitionModuleLink" containment="true"
                resolveProxies="false"/>
43    </eClassifiers>
44    <eClassifiers xsi:type="ecore:EClass" name="ExhibitionModule">
45      <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
46      <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
47      <eStructuralFeatures xsi:type="ecore:EReference" name="sequences"
            upperBound="-1"
48          eType="#//Sequence" containment="true" resolveProxies="false"/>
49      <eStructuralFeatures xsi:type="ecore:EReference" name="startSequence"
            eType="#//Sequence"/>
50      <eStructuralFeatures xsi:type="ecore:EReference" name="category" eType=
            "#//Category"
51          containment="true" resolveProxies="false"/>
52      <eStructuralFeatures xsi:type="ecore:EReference" name="slides"
            upperBound="-1"
53          eType="#//Slide" containment="true" resolveProxies="false"/>
54      <eStructuralFeatures xsi:type="ecore:EAttribute" name="
            exhibitionModuleId" eType="ecore:EDataType http://www.eclipse.org/
            emf/2002/Ecore#//EString"/>
55    </eClassifiers>
56    <eClassifiers xsi:type="ecore:EClass" name="ExhibitionModuleLink"
          eSuperTypes="#//Link">
57      <eStructuralFeatures xsi:type="ecore:EReference" name="
```

```
        exhibitionModuleTarget"
58      eType="#//ExhibitionModule"/>
59    </eClassifiers>
60    <eClassifiers xsi:type="ecore:EClass" name="Image" eSuperTypes="#//Media"
          >
61      <eStructuralFeatures xsi:type="ecore:EAttribute" name="imagePath" eType
            ="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/
            >
62      <eStructuralFeatures xsi:type="ecore:EAttribute" name="width" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
63      <eStructuralFeatures xsi:type="ecore:EAttribute" name="height" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
64    </eClassifiers>
65    <eClassifiers xsi:type="ecore:EClass" name="Link">
66      <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
67      <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
68      <eStructuralFeatures xsi:type="ecore:EAttribute" name="linkId" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
69    </eClassifiers>
70    <eClassifiers xsi:type="ecore:EClass" name="MapSceneLink" eSuperTypes="
          #//Link">
71      <eStructuralFeatures xsi:type="ecore:EReference" name="source" eType="
            #//ExhibitionMap"/>
72      <eStructuralFeatures xsi:type="ecore:EReference" name="target" eType="
            #//Scene"/>
73    </eClassifiers>
74    <eClassifiers xsi:type="ecore:EClass" name="Media">
75      <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
76      <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
77    </eClassifiers>
78    <eClassifiers xsi:type="ecore:EClass" name="Scene">
79      <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
80      <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
81      <eStructuralFeatures xsi:type="ecore:EReference" name="backgroundImage"
             eType="#//Image"
82          containment="true" resolveProxies="false"/>
83      <eStructuralFeatures xsi:type="ecore:EReference" name="links"
            upperBound="-1"
84          eType="#//Link" containment="true" resolveProxies="false"/>
85      <eStructuralFeatures xsi:type="ecore:EAttribute" name="sceneId" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
86    </eClassifiers>
```

```
87  <eClassifiers xsi:type="ecore:EClass" name="SceneLink" eSuperTypes="#//
        Link">
88    <eStructuralFeatures xsi:type="ecore:EReference" name="sceneLinkTarget"
          eType="#//Scene"/>
89  </eClassifiers>
90  <eClassifiers xsi:type="ecore:EClass" name="Sequence">
91    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
92    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
          eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
          EString"/>
93    <eStructuralFeatures xsi:type="ecore:EAttribute" name="slideOrder"
          eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
          EString"/>
94    <eStructuralFeatures xsi:type="ecore:EAttribute" name="sequenceId"
          eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
          EString"/>
95  </eClassifiers>
96  <eClassifiers xsi:type="ecore:EClass" name="Slide">
97    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
98    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
          eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
          EString"/>
99    <eStructuralFeatures xsi:type="ecore:EAttribute" name="subTitle" eType=
          "ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
100   <eStructuralFeatures xsi:type="ecore:EReference" name="audio"
          upperBound="-1"
101       eType="#//Audio" containment="true" resolveProxies="false"/>
102   <eStructuralFeatures xsi:type="ecore:EReference" name="texts"
          upperBound="-1"
103       eType="#//Text" containment="true" resolveProxies="false"/>
104   <eStructuralFeatures xsi:type="ecore:EReference" name="video"
          upperBound="-1"
105       eType="#//Video" containment="true" resolveProxies="false"/>
106   <eStructuralFeatures xsi:type="ecore:EReference" name="images"
          upperBound="-1"
107       eType="#//Image" containment="true" resolveProxies="false"/>
108   <eStructuralFeatures xsi:type="ecore:EAttribute" name="footer" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
109   <eStructuralFeatures xsi:type="ecore:EAttribute" name="slideTemplateId"
          eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
          EString"/>
110   <eStructuralFeatures xsi:type="ecore:EAttribute" name="slideId" eType="
          ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
111 </eClassifiers>
112 <eClassifiers xsi:type="ecore:EClass" name="SlideTemplate">
113   <eStructuralFeatures xsi:type="ecore:EAttribute" name="slideTemplateId"
          eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
          EString"/>
114   <eStructuralFeatures xsi:type="ecore:EAttribute" name="title" eType="
```

```
        ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
115     <eStructuralFeatures xsi:type="ecore:EAttribute" name="description"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
116     <eStructuralFeatures xsi:type="ecore:EAttribute" name="templateFilePath
            " eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
117     <eStructuralFeatures xsi:type="ecore:EAttribute" name="templateType"
            eType="#//TemplateType"/>
118   </eClassifiers>
119   <eClassifiers xsi:type="ecore:EClass" name="Text" eSuperTypes="#//Media">
120     <eStructuralFeatures xsi:type="ecore:EAttribute" name="textFilePath"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
            EString"/>
121     <eStructuralFeatures xsi:type="ecore:EAttribute" name="text" eType="
            ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
122   </eClassifiers>
123   <eClassifiers xsi:type="ecore:EClass" name="Video" eSuperTypes="#//Media"
        >
124     <eStructuralFeatures xsi:type="ecore:EAttribute" name="videoFile" eType
            ="#//File"/>
125   </eClassifiers>
126   <eClassifiers xsi:type="ecore:EDataType" name="File" instanceClassName="
        java.io.File"/>
127   <eClassifiers xsi:type="ecore:EDataType" name="TemplateType"
        instanceClassName="org.jat.exhibition.TemplateType"/>
128 </ecore:EPackage>
```

Listing D-1 Ausschnitt der Ecore-Datei zur Definition des Metamodells

D.2.2. Editor zur Bearbeitung des Metamodells

Abbildung D-2 zeigt den EMF-Editor, in dem das Metamodell bearbeitet werden kann.

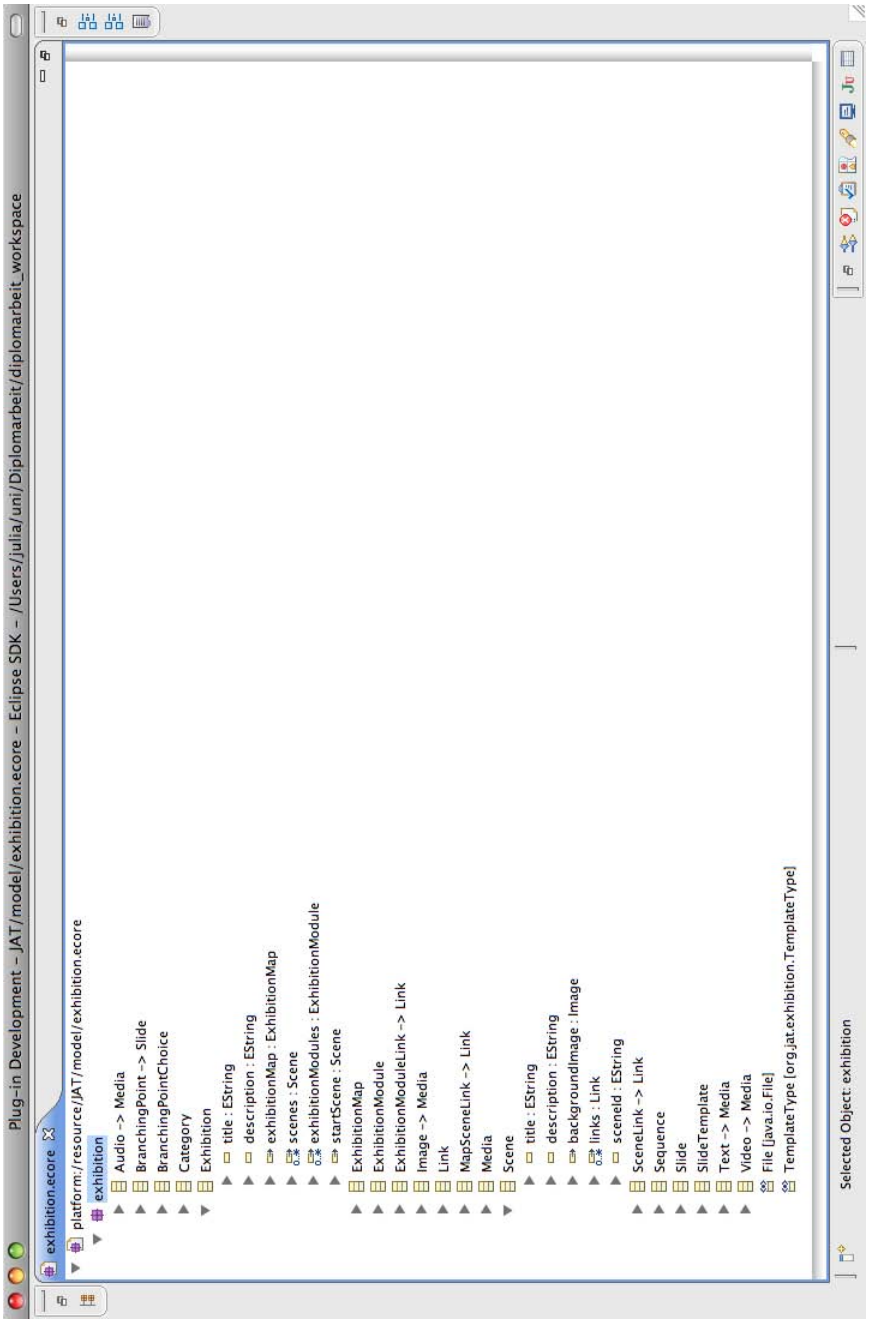


Abb. D-2. Metamodell des Prototyps im EMF-Editor

D.2.3. Genmodel von EMF

Abbildung D-3 zeigt das Genmodel von EMF für das Metamodell.

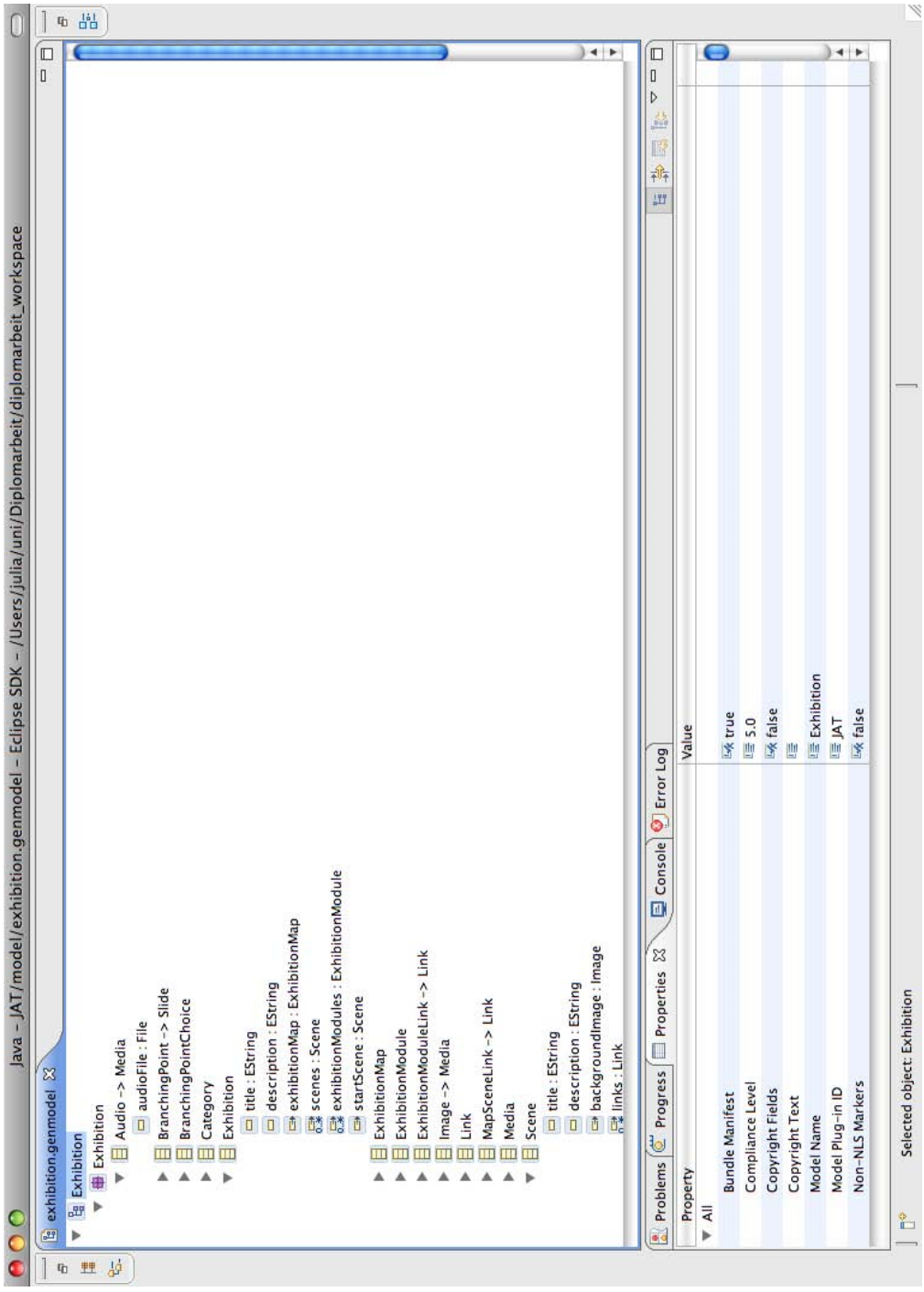


Abb. D-3. Genmodel für das Metamodell

D.3. Ergänzungen zur Entwicklung des GMF-Modellierungstools

D.3.1. Graphical Definition Model

Abbildung D-4 zeigt das Graphical Definition Model für das zu entwickelnde Modellierungstool. Der Properties-View zeigt die Eigenschaften für das graphische Elemente, welches eine Scene repräsentiert. Das Graphical Definition Model befindet sich im XMI-Format auf der Diplomarbeit beiliegenden CD.

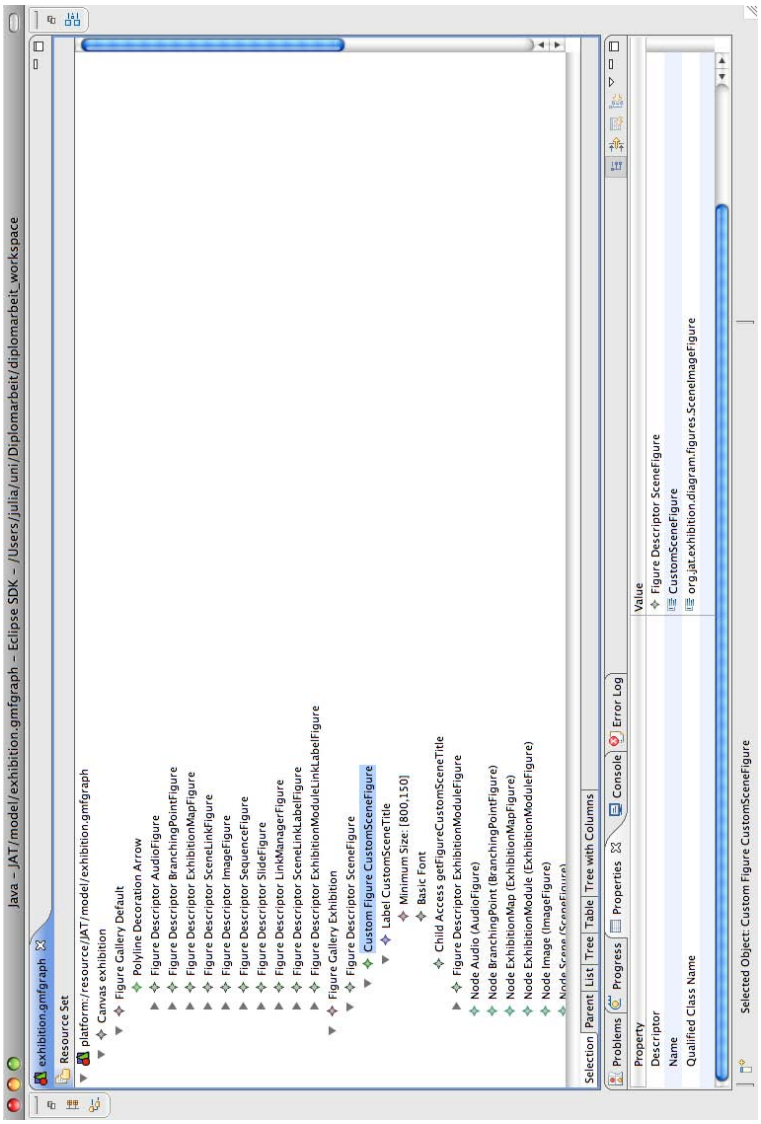


Abb. D-4. Graphical Definition Model des GMF-Modellierungstool

D.3.2. Ergänzungen zur Entwicklung einer konkreten Syntax

Nachfolgend finden sich Bilder der Elemente der konkreten Syntax der DSL.

Abb. D-5. Scene

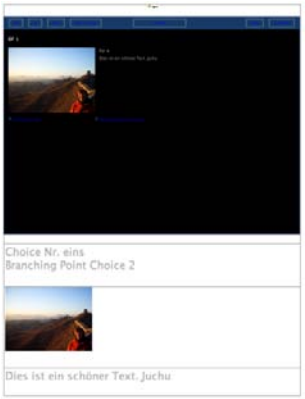
Scene, Scene Link und Exhibition Module Link
Abbildung D-5 zeigt die Darstellung einer Scene. Der Scene wurden ein Scene Link und eine Exhibition Module Link hinzugefügt. Sie werden durch die weißen Rechtecke auf dem Bild der Scene repräsentiert.

Abb. D-6. Exhibition Module

Exhibition Module
Abbildung D-6 zeigt das Symbol eines Netzwerks. Dieses Symbol repräsentiert eine Exhibition Module im Modell. Das Netzwerk soll den netzwerkartigen Aufbau eines Exhibition Modules durch Sequences darstellen.

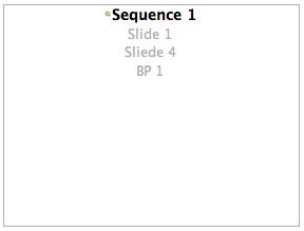
Abb. D-7. Slide

Slide
Abbildung D-7 zeigt die Darstellung eines Slides. Die Darstellung ist dreigeteilt. Im oberen Teil befindet sich eine Vorschau der Webseite, die für den Slide generiert wird. Darunter befindet sich jeweils eine Übersicht über alle Bilder und alle Texte, die dem Slide hinzugefügt wurden.



Branching Point
Abbildung D-8 zeigt die Darstellung eines Branching Points. Die Darstellung ist viergeteilt. Im oberen Teil befindet sich eine Vorschau der Webseite, die für diesen Branching Point generiert wird. Darunter befindet sich ein Abschnitt, in dem die Branching Point Choices, die diesem Branching Point hinzugefügt wurden, angezeigt werden. Darunter ist jeweils ein Abschnitt für die Bilder und die Texte, die in diesem Branching Point enthalten sind.

Abb. D-8. Branching Point



Sequence
Abbildung D-9 zeigt die Darstellung einer Sequence. Innerhalb des Rechtecks befindet sich eine Liste mit den Titeln der Slides und Branching Point, auf die die Sequence verweist.

Abb. D-9. Sequence

Die Darstellung von Bildern und Texte ist eine Anzeige des entsprechenden Bilds beziehungsweise des Textes. Eine Virtual Exhibition hat keine graphische Anzeige.

Abbildung D-10 zeigt den Zusammenhang der Elemente der konkreten Syntax zu den generierten Webseiten.

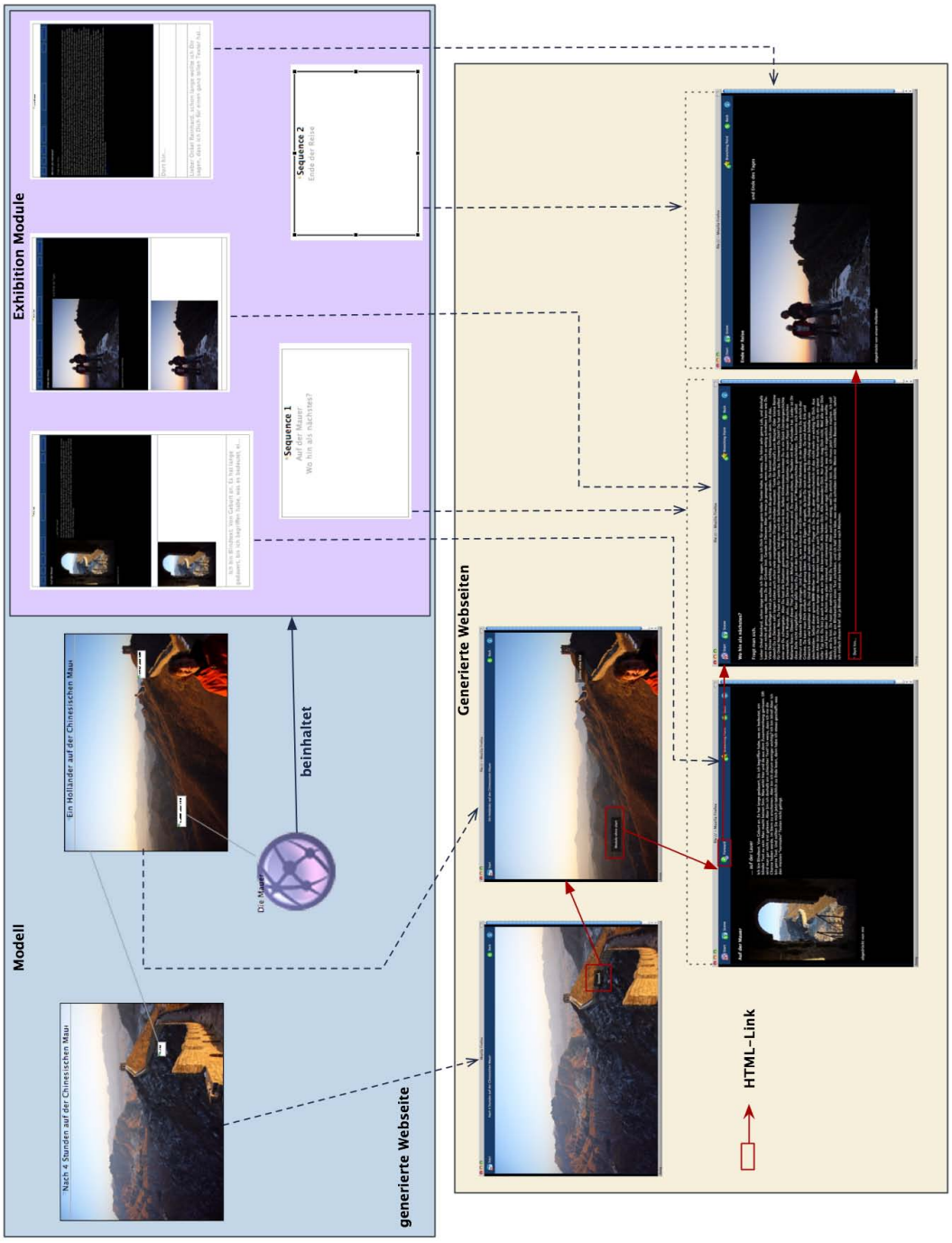


Abb. D-10. Elemente der konkreten Syntax und deren Darstellung in den generierten Webseiten

D.3.3. Darstellung der Navigation durch die Scenes

Abbildung D-11 zeigt einen Screenshot des Modellierungstools. Die Verbindungen zu den Scenes und Exhibition Modules von den Scene Links und Exhibition Module Links werden durch Linien dargestellt. Dies ermöglicht den Überblick über die Navigation durch eine virtuelle Ausstellung.

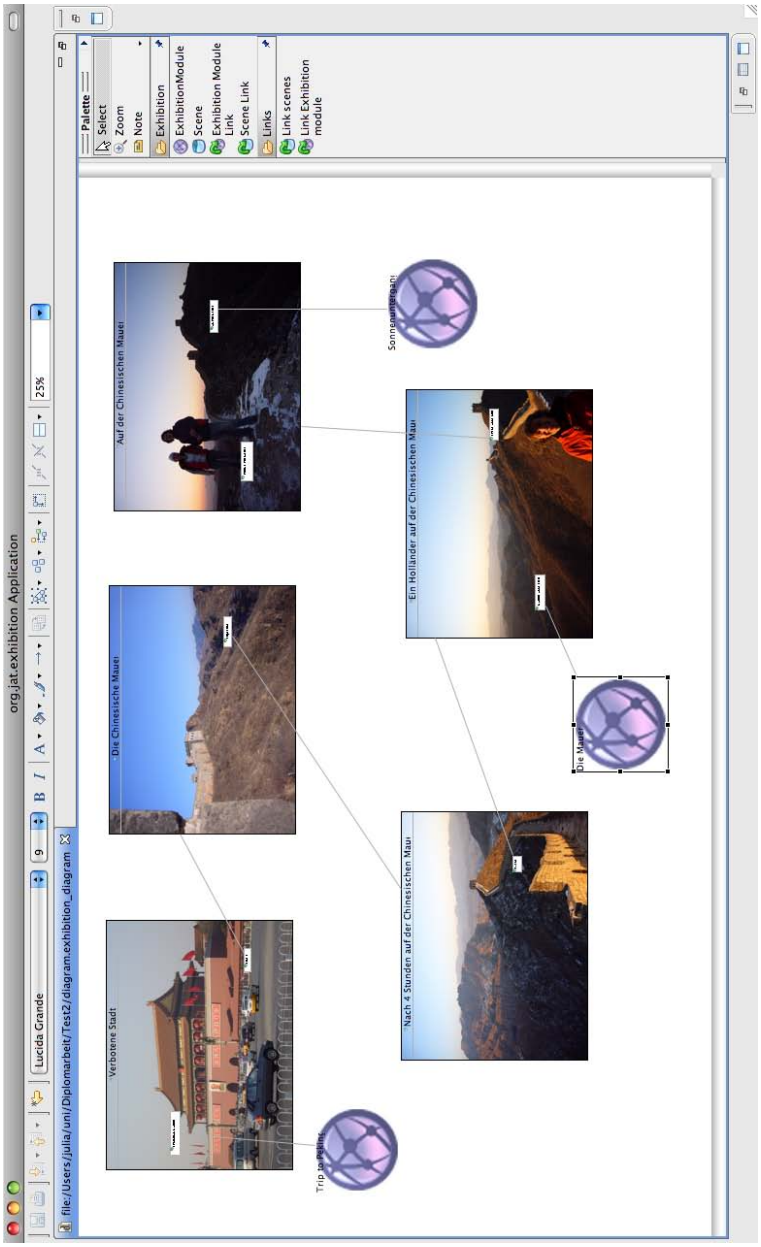


Abb. D-11. Navigation durch eine virtuelle Ausstellung

D.3.4. Tooling Definition Model

Abbildung D-12 zeigt das Tooling Definition Model für das zu entwickelnden GMF-Modellierungstool. Der Properties-View zeigt die Eigenschaften für die zweite Gruppe von Palleteneinträgen.

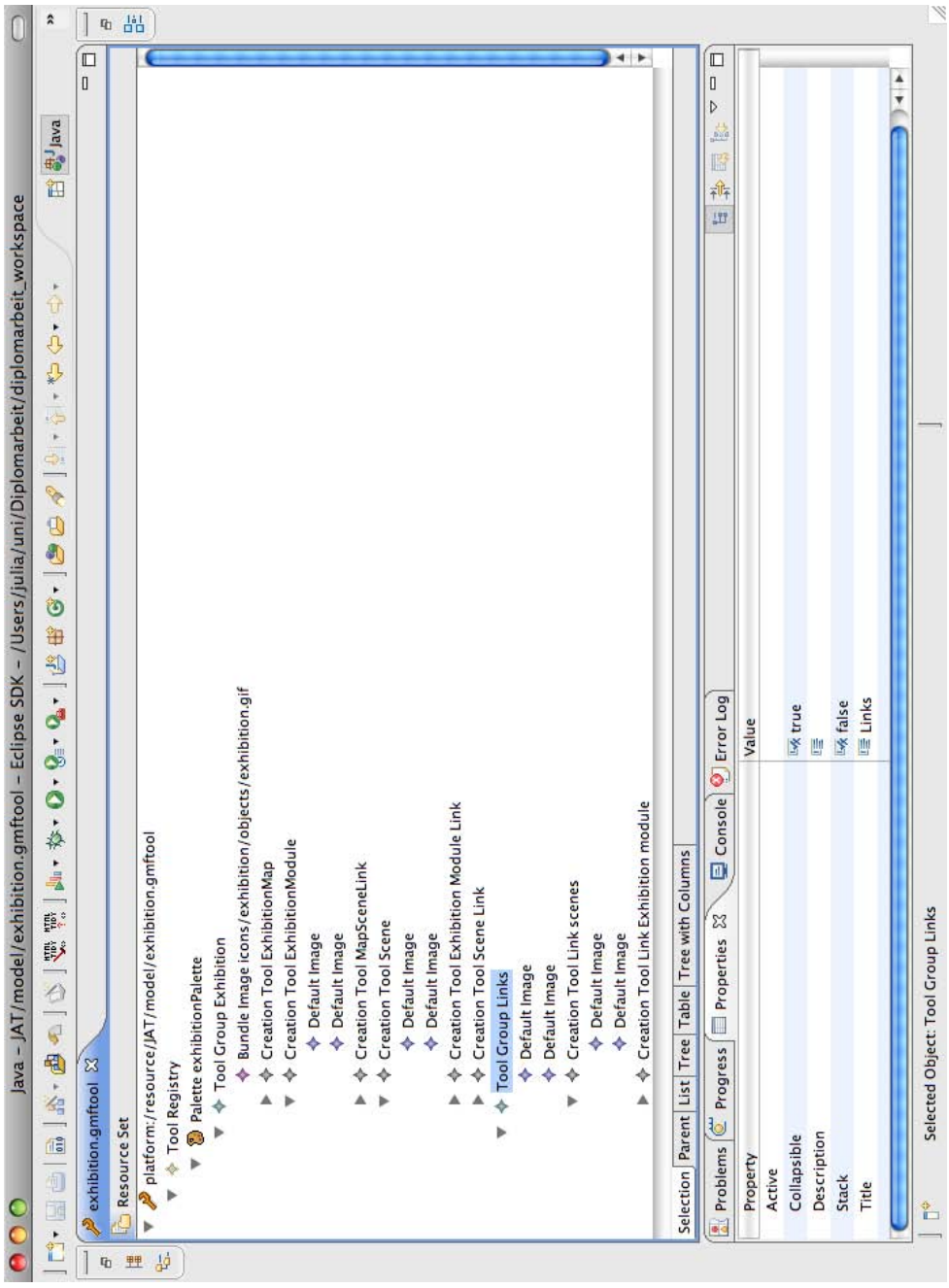


Abb. D-12. Tooling Definition Model des Modellierungstools

D.3.5. Aufbau des Modellierungstools

Abbildung D-13 zeigt den Aufbau des Modellierungstools, das als Prototyp entwickelt wurde.

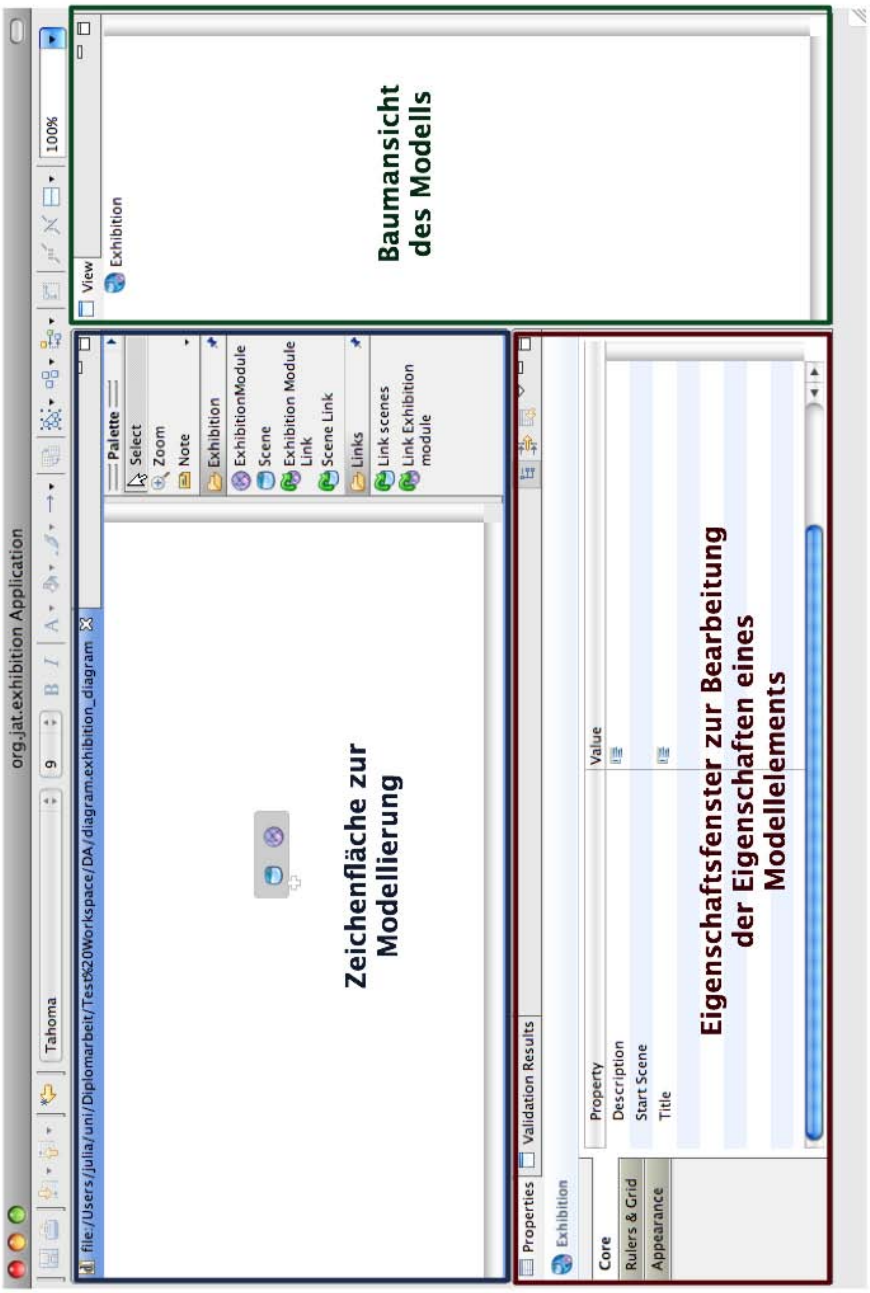


Abb. D-13. Aufbau des GMF-Modellierungstools

D.3.6. Menüleisten des Modellierungstools

Abbildung D-14 zeigt einen Screenshot des Modellierungstools. Am oberen Bildrand ist das Hauptmenü zu sehen. Am oberen Rand des Anwendungsfensters des Modellierungstools ist die Coolbar zu finden.

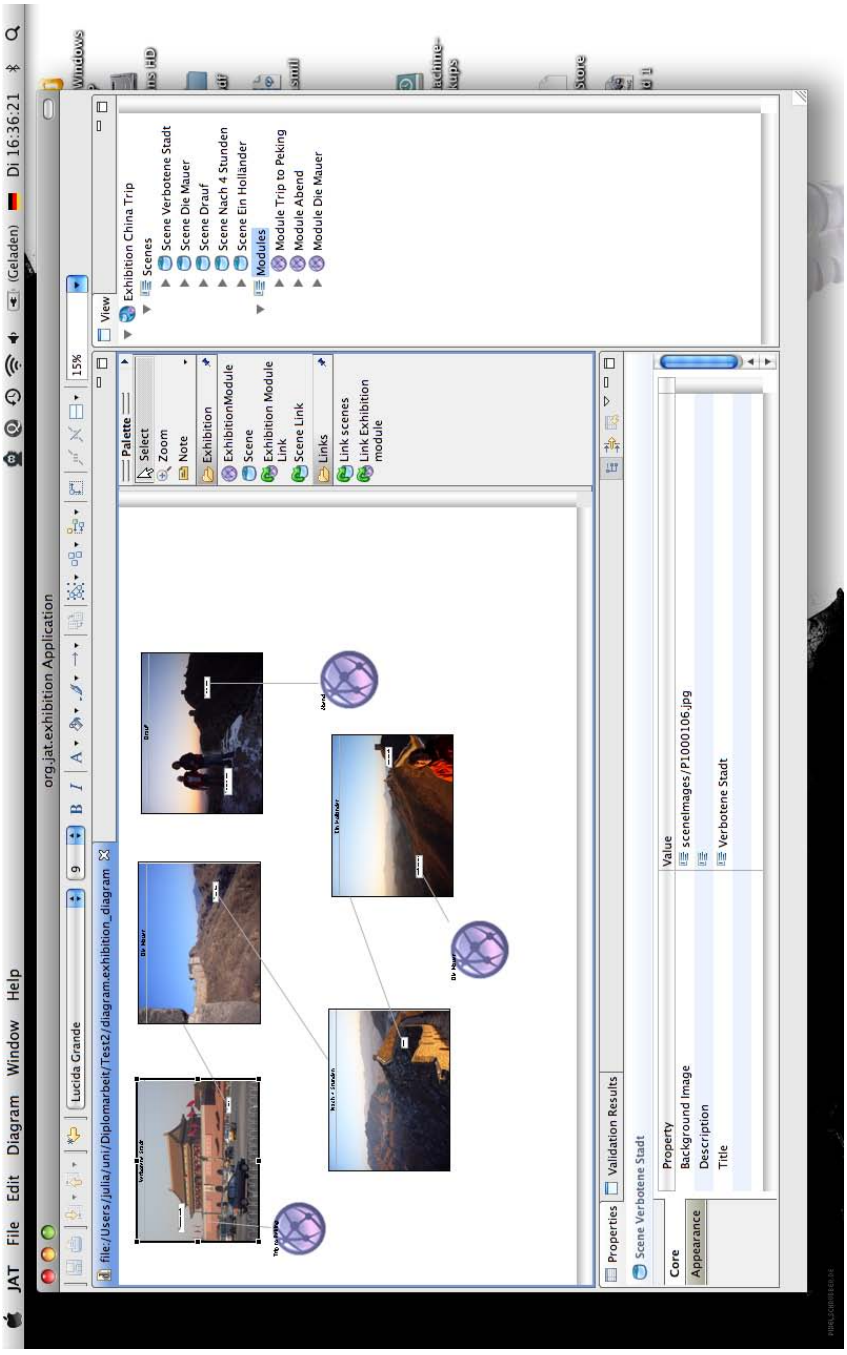


Abb. D-14. Modellierungstool mit zwei Menüleisten

D.3.7. Validierung

Abbildung D-15 zeigt das Modellierungstool nach der Validierung des Modells. Die Elemente, die gegen einen Constraint verstoßen sind durch Icons gekennzeichnet.

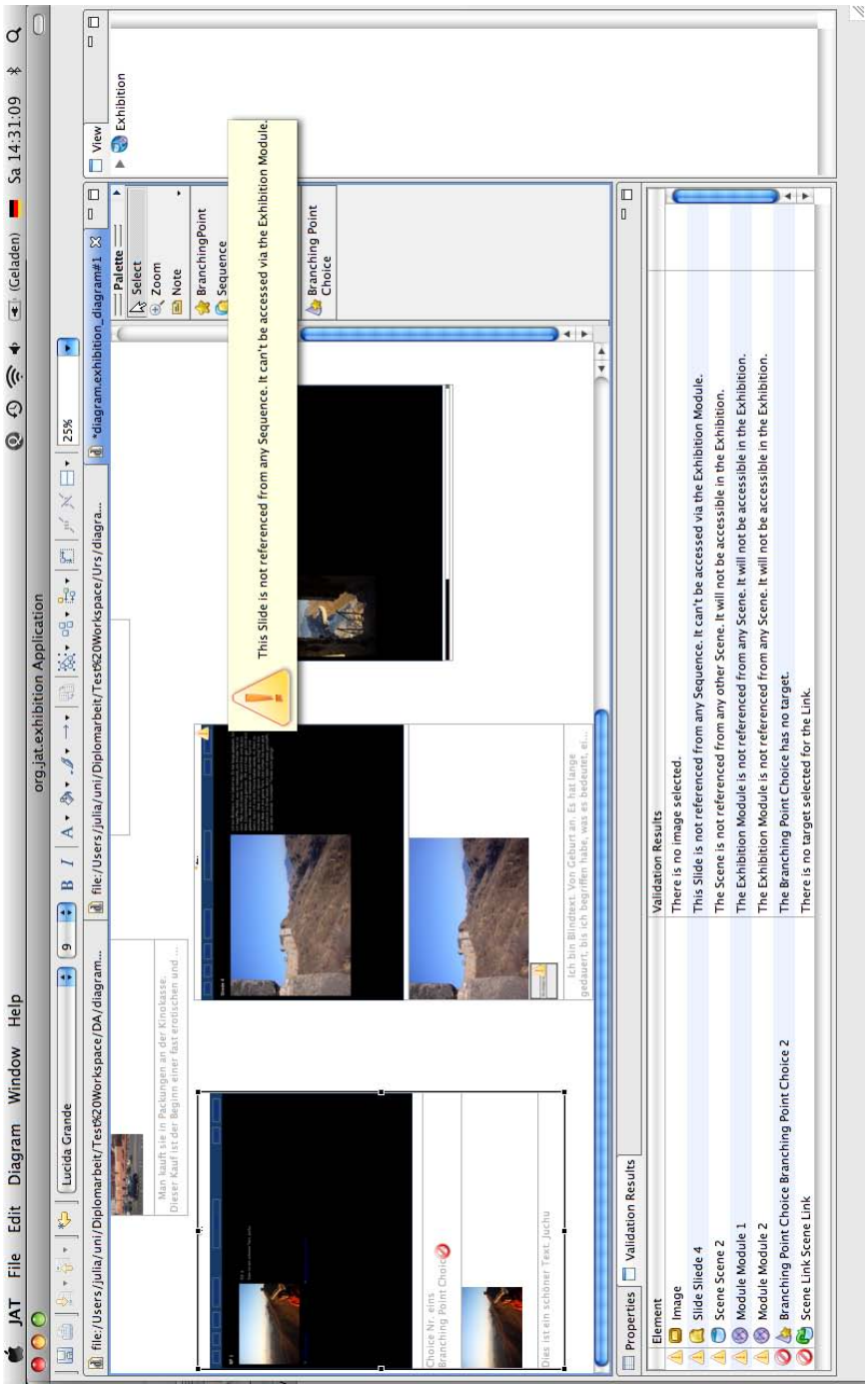


Abb. D-15. Validierung im GMF-Modellierungstool

D.3.8. Slide Templates

Listing D-2 zeigt ein JAT Page Template für einen Branching Point, das ein Bild und rechts daneben einen Text anzeigt.

```
<!--
   Id:BPTemplate01
   Title: Branching Point Template 1 (1 image, 1 text block)
   Description: 1 image on left, 1 text block on the right side.
   CSS:bptemplate01.css
 -->
 <div id="bptemplate01">
    <div id="title" class="title"></div>
    <div id="image01" class="image"></div>
    <div id="subtitle" class="subtitle"></div>
    <div id="text" class="text"></div>
    <div id="choices" class="choices"></div>
    <div id="footer" class="footer"></div>
 </div>
```

Listing D-2 JAT Page Template

Listing D-3 zeigt die zu dem JAT Page Template gehörende CSS-Datei.

```
#bptemplate01 #subtitle {
}

#bptemplate01 #text {
}


#bptemplate01 #image01 {
    float: left;
}

#bptemplate01 #choices {
    clear: left;
}

#bptemplate01 #footer {
   clear:both;
}
```

Listing D-3 CSS-Datei für die Definition eines Slide Templates

D.3.9. Mehrere geöffnete Editoren

Abbildung D-16 zeigt die Darstellung von mehreren geöffneten Editoren durch Reiter.

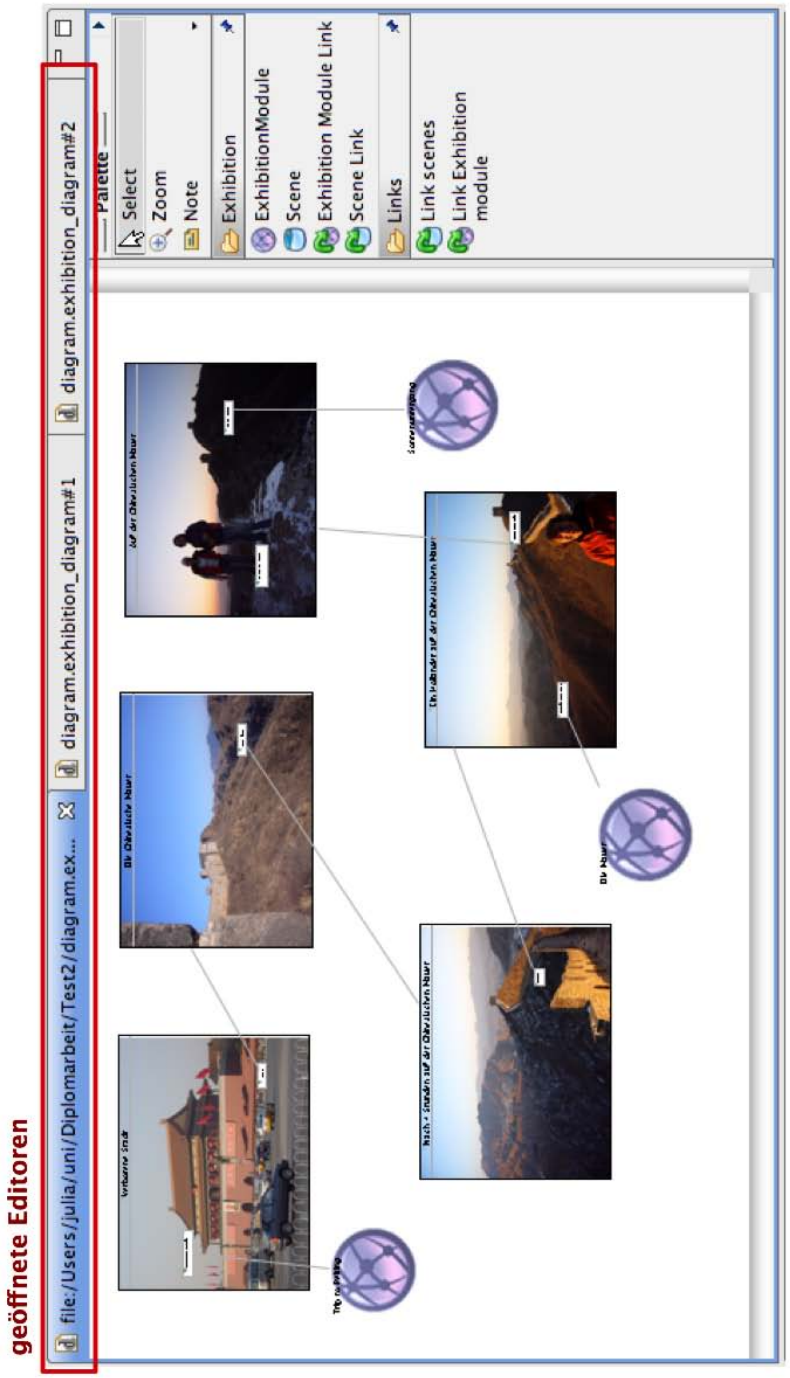


Abb. D-16. Mehrere geöffnete Editoren

D.4. Ergänzungen zur Entwicklung des Codegenerators

D.4.1. Template zur Generierung von Webseiten für Exhibition Modules

Listing D-4 zeigt das Template für die Generierung von Webseiten für die Exhibition Modules einer virtuellen Ausstellung.

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<!--
   Icons of buttons back to scene, back to branching point and back:
   Copyright by Miloszwl.com
   Contact: miloszwl@miloszwl.com
   Web: miloszwl.deviantart.com
-->
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<link id="maincss" rel="stylesheet" type="text/css" href="@%css%@" />
<link id="templatecss" rel="stylesheet" type="text/css" href="@%templatecss
    %@" />
<script id="navigationJs" type="text/javascript">@%navJs%@</script>
</head>
<body onLoad="setForwardVisibility()">
<table cellpadding="0" cellspacing="0" id="base">
  <tr>
    <td align="center">
    <div id="navigation">
      <div class="nav_item_left_image"><a class="start" id="start" href="@%
          start%@">@%start_title%@</a></div>

      <div class="nav_item_left_image"><a id="backToRoom" href="@%
          backToRoom%@">@%back_scene_title%@</a></div>
        <div class="nav_item_next_image" id="forward_div"><a id="forward"
            href="javascript:window.location.href=nextSlide()">@%
            forward_title%@</a></div>
        <div class="nav_item_right_image"><a id="copyright" href="@%
            copyright%@">@%copyright_title%@</a></div>
      <div class="nav_item_right_image"><a id="back" href="@%back_link%@">@
          %back_title%@</a></div>
      <div class="nav_item_right_image"><a id="backBP" href="javascript:
          window.location.href=branchingPoint()">@%back_bp_title%@</a></div
          >
    </div>
    <div id="body">
      <div id="exhibitionmodule">@%content%@</div>
      </div>
    </td>
  </tr>
</table>
```

```
34 </body>
35 </html>
```

Listing D-4 JAT Generation Template für Exhibition Modules

D.4.2. Quelltextauszug der Implementierung des Ersetzungsprozesses

Listing D-5 zeigt einen Auszug aus der Methode zur Generierung einer Datei aus einem Template. Das Template wird Zeile für Zeile eingelesen. Für jede Zeile werden die Tags durch entsprechende Werte ersetzt. Im Anschluss werden nicht verwendete Tags aus der Zeile gelöscht. Die Zeilen werden über einen `BufferedWriter` in eine Datei geschrieben.

```
1  while ((line = bufferedReader.readLine()) != null)
2  {
3     String replacedLine = line;
4     if ((variables != null) || (variables.size() > 0))
5     {
6        for (String key : variables.keySet())
7        {
8           String var = key;
9           if (!var.startsWith(VariablesMarker.BEGIN.getMarker()) && !var.
                endsWith(VariablesMarker.END.getMarker()))
10          var = VariablesMarker.BEGIN.getMarker() + var + VariablesMarker.
                END.getMarker();
11          if(replacedLine.contains(var))
12             replacedLine = replacedLine.replace(var, variables.get(key));
13       }
14    }
15    Pattern pattern = Pattern.compile(VariablesMarker.BEGIN.getMarker() + "
          .*?" + VariablesMarker.END.getMarker());
16    Matcher matcher = pattern.matcher(replacedLine);
17    while (matcher.find())
18    {
19       String found = matcher.group();
20       replacedLine = replacedLine.replace(found, "");
21    }
22    bufferedWriter.append(replacedLine + "\n");
23 }
24 bufferedReader.close();
25 bufferedWriter.close();
```

Listing D-5 Quelltextauszug der Implementierung des Ersetzungsprozesses

D.5. Ergänzungen zu den verwendeten Entwurfsmustern

D.5.1. Quelltext der Klasse `JatGenerationEngineFactory`

Listing D-6 zeigt den Quelltext der Klasse `JatGenerationEngineFactory`. Diese Klasse ist Teil der Umsetzung des Abstract Factory-Pattern im Zusammenhang mit dem Strategy-Pattern.

```
1  package org.jat.generation.engine.services;
2
3  import org.jat.generation.engine.internal.JatHtmlGeneratorEngineImpl;
4
5  /**
6   * Factory for retrieving generator dependent on the generation type.
7   * @author Julia Damerow
8   */
9  public class JatGenerationEngineFactory {
10
11    private static JatGenerationEngineFactory instance = null;
12    private JatGenerationEngineFactory() { }
13
14    /**
15     * Get instance of JatGenerationFactory.
16     * @return Instance of JatGenerationFactory
17     */
18    public static JatGenerationEngineFactory getInstance()
19    {
20       if (instance == null)
21          instance = new JatGenerationEngineFactory();
22
23       return instance;
24    }
25
26    /**
27     * Get genertor.
28     * @param type Type of generator
29     * @param outputFolder Path to output folder.
30     * @return JatGenerator for given type.
31     */
32    public JatGeneratorEngine getGenerator(JatGenerationEngineType type,
          String outputFolder)
33    {
34       if (type == JatGenerationEngineType.HTML)
35          return new JatHtmlGeneratorEngineImpl(outputFolder);
36
37       return null;
38    }
39 }
```

Listing D-6 Quelltext der Klasse `JatGenerationEngineFactory`

D.5.2. Quelltext zur Umsetzung des Singleton-Patterns

Listing D-7 zeigt eine gekürzte Fassung des Quelltexts der Klasse `WorkspaceManager`. Diese setzt das Singleton-Pattern um.

```
1  package org.jat.workspace.services;
2
3  import java.io.File;
4  import java.util.UUID;
5
6  public class WorkspaceManager {
7
8     private static WorkspaceManager instance = null;
9
10    private String workspacePath;
11    private String imageFolderPath;
12    private String exhibitionModuleImageFolderPath;
13    private String htmlFolderPath;
14    private String htmlSceneImageFolderPath;
15    private String htmlExhibitionModuleImageFolderPath;
16
17    private WorkspaceManager() {
18       imageFolderPath = "sceneImages";
19       exhibitionModuleImageFolderPath = "exhibitionModuleImages";
20       htmlFolderPath = File.separator + "html";
21       htmlSceneImageFolderPath = "sceneImages";
22       htmlExhibitionModuleImageFolderPath = "exhibitionModuleImages";
23    }
24
25    public static WorkspaceManager getInstance() {
26       if (instance == null)
27          instance = new WorkspaceManager();
28       return instance;
29    }
30
31    public String getWorkspacePath() {
32       return workspacePath;
33    }
34
35    public void setWorkspacePath(String workspacePath) {
36       workspacePath = workspacePath.replace("%20", " ");
37       this.workspacePath = workspacePath;
38
39       // create image folder for scene images
40       File imageFolderFile = new File(workspacePath + File.separator
41             + imageFolderPath);
42       if (!imageFolderFile.exists()) {
43          imageFolderFile.mkdir();
44       } else if (imageFolderFile.isFile()) {
45          imageFolderPath = UUID.randomUUID().toString();
46          imageFolderFile.mkdir();
```

```
47        }
48
49        // create image folder for exhibition modules
50        File exhibitionModuleImageFile = new File(workspacePath
51              + File.separator + exhibitionModuleImageFolderPath);
52        if (!exhibitionModuleImageFile.exists()) {
53           exhibitionModuleImageFile.mkdir();
54        } else if (exhibitionModuleImageFile.isFile()) {
55           exhibitionModuleImageFolderPath = UUID.randomUUID().toString();
56           exhibitionModuleImageFile.mkdir();
57        }
58
59        // create folder for generated html pages
60        File htmlFolderFile = new File(workspacePath + htmlFolderPath);
61        if (!htmlFolderFile.exists()) {
62           htmlFolderFile.mkdir();
63        } else if (htmlFolderFile.isFile()) {
64           htmlFolderPath = UUID.randomUUID().toString();
65           htmlFolderFile.mkdir();
66        }
67
68        // create folder for scene images for generated html pages
69        File htmlSceneImageFolder = new File(workspacePath + htmlFolderPath
70              + File.separator + imageFolderPath);
71        if (!htmlSceneImageFolder.exists()) {
72           htmlSceneImageFolder.mkdir();
73        } else if (htmlSceneImageFolder.isFile()) {
74           htmlSceneImageFolderPath = UUID.randomUUID().toString();
75           htmlSceneImageFolder.mkdir();
76        }
77
78        // create folder for exhibition module images for generated html
              pages
79        File htmlExhibitionModuleImageFolder = new File(workspacePath
80              + htmlFolderPath + File.separator
81              + exhibitionModuleImageFolderPath);
82        if (!htmlExhibitionModuleImageFolder.exists()) {
83           htmlExhibitionModuleImageFolder.mkdir();
84        } else if (htmlExhibitionModuleImageFolder.isFile()) {
85           htmlExhibitionModuleImageFolderPath = UUID.randomUUID().toString()
                 ;
86           htmlExhibitionModuleImageFolder.mkdir();
87        }
88     }
89
90     /* A lot of getter and setter methods... */
91  }
```

Listing D-7 Quelltext zur Umsetzung des Singleton-Patterns

D.6. Ergänzungen zu den Implementierungsdetails

D.6.1. Quelltext der `plugin.xml` des Hauptprojekts

Listing D-8 zeigt den Quelltext der `plugin.xml` des Hauptprojekts.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
   <extension point="org.eclipse.team.core.fileTypes">
      <?gmfgen generated="true"?>
      <fileTypes
         type="text"
         extension="exhibition_diagram">
      </fileTypes>
   </extension>

   <extension point="org.eclipse.emf.ecore.extension_parser">
      <?gmfgen generated="true"?>
      <parser
         type="exhibition_diagram"
         class="org.eclipse.gmf.runtime.emf.core.resources.
             GMFResourceFactory">
      </parser>
   </extension>

   <extension point="org.eclipse.ui.editors">
      <?gmfgen generated="true"?>
      <editor
         id="org.jat.exhibition.diagram.part.ExhibitionDiagramEditorID"
         name="%editorName"
         icon="icons/obj16/ExhibitionDiagramFile.gif"
         extensions="exhibition_diagram"
         default="true"
         class="org.jat.exhibition.diagram.part.ExhibitionDiagramEditor"
         matchingStrategy="org.jat.exhibition.diagram.part.
             ExhibitionMatchingStrategy"
         contributorClass="org.jat.exhibition.diagram.part.
             ExhibitionDiagramActionBarContributor">
      </editor>
   </extension>

   <extension point="org.eclipse.ui.contexts">
      <?gmfgen generated="true"?>
      <context
         description="%context.description"
         id="org.jat.exhibition.diagram.ui.diagramContext"
         name="%context.name"
         parentId="org.eclipse.gmf.runtime.diagram.ui.diagramContext">
      </context>
```

```
42    </extension>
43
44    <extension point="org.eclipse.ui.newWizards">
45       <?gmfgen generated="true"?>
46      <wizard
47         name="%newWizardName"
48         icon="icons/obj16/ExhibitionDiagramFile.gif"
49         category="org.eclipse.ui.Examples"
50         class="org.jat.exhibition.diagram.part.ExhibitionCreationWizard"
51         id="org.jat.exhibition.diagram.part.ExhibitionCreationWizardID">
52        <description>%newWizardDesc</description>
53       </wizard>
54    </extension>
55
56    <extension point="org.eclipse.ui.popupMenus">
57       <?gmfgen generated="true"?>
58       <objectContribution
59             adaptable="false"
60             id="org.jat.exhibition.diagram.LoadResource"
61             objectClass="org.jat.exhibition.diagram.edit.parts.
                   ExhibitionEditPart">
62          <action
63                class="org.jat.exhibition.diagram.part.
                      ExhibitionLoadResourceAction"
64                enablesFor="1"
65                id="org.jat.exhibition.diagram.LoadResourceAction"
66                label="%loadResourceActionLabel"
67                menubarPath="additions">
68          </action>
69       </objectContribution>
70    </extension>
71
72       <extension point="org.eclipse.ui.actionSets">
73       <?gmfgen generated="false"?>
74       <!--<actionSet
75             label="%initDiagramActionLabel"
76             visible="true"
77             id="org.jat.exhibition.diagram.InitDiagram">
78          <action
79                label="%initDiagramActionLabel"
80                class="org.jat.exhibition.diagram.part.
                      ExhibitionInitDiagramFileAction"
81                menubarPath="file/additions"
82                id="org.jat.exhibition.diagram.InitDiagramAction">
83          </action>
84       </actionSet>-->
85    </extension>
86
87    <extension point="org.eclipse.gmf.runtime.common.ui.services.action.
          globalActionHandlerProviders">
88       <?gmfgen generated="true"?>
```

```
89      <GlobalActionHandlerProvider
90         class="org.eclipse.gmf.runtime.diagram.ui.providers.
               DiagramGlobalActionHandlerProvider"
91         id="ExhibitionPresentation">
92         <Priority name="Lowest"/>
93         <ViewId id="org.jat.exhibition.diagram.part.
               ExhibitionDiagramEditorID">
94            <ElementType class="org.eclipse.gmf.runtime.diagram.ui.
                  editparts.IGraphicalEditPart">
95               <GlobalActionId actionId="delete"/>
96            </ElementType>
97            <ElementType class="org.eclipse.gmf.runtime.diagram.ui.
                  editparts.DiagramEditPart">
98               <GlobalActionId actionId="save"/>
99            </ElementType>
100        </ViewId>
101     </GlobalActionHandlerProvider>
102     <GlobalActionHandlerProvider
103           class="org.eclipse.gmf.runtime.diagram.ui.render.providers.
                  DiagramUIRenderGlobalActionHandlerProvider"
104           id="ExhibitionRender">
105        <Priority name="Lowest"/>
106        <ViewId id="org.jat.exhibition.diagram.part.
               ExhibitionDiagramEditorID">
107           <ElementType class="org.eclipse.gmf.runtime.diagram.ui.
                  editparts.IGraphicalEditPart">
108              <GlobalActionId actionId="cut"/>
109              <GlobalActionId actionId="copy"/>
110              <GlobalActionId actionId="paste"/>
111           </ElementType>
112        </ViewId>
113     </GlobalActionHandlerProvider>
114  </extension>
115
116  <extension point="org.eclipse.gmf.runtime.common.ui.services.action.
        contributionItemProviders">
117     <?gmfgen generated="true"?>
118     <contributionItemProvider
119           class="org.eclipse.gmf.runtime.diagram.ui.providers.
                  DiagramContributionItemProvider"
120           checkPluginLoaded="false">
121        <Priority name="Low"/>
122        <popupContribution class="org.eclipse.gmf.runtime.diagram.ui.
               providers.DiagramContextMenuProvider">
123           <popupStructuredContributionCriteria objectClass="org.jat.
                  exhibition.diagram.edit.parts.
                  ExhibitionModuleLinkExhibitionModuleTargetEditPart"/>
124           <popupAction path="/editGroup" id="deleteFromModelAction"/>
125           <popupPredefinedItem id="deleteFromDiagramAction" remove="true"
                  />
126        </popupContribution>
```

```
127         <popupContribution class="org.eclipse.gmf.runtime.diagram.ui.
                providers.DiagramContextMenuProvider">
128            <popupStructuredContributionCriteria objectClass="org.jat.
                   exhibition.diagram.edit.parts.
                   SceneLinkSceneLinkTargetEditPart"/>
129            <popupAction path="/editGroup" id="deleteFromModelAction"/>
130            <popupPredefinedItem id="deleteFromDiagramAction" remove="true"
                   />
131         </popupContribution>
132      </contributionItemProvider>
133   </extension>
134
135   <extension point="org.eclipse.core.runtime.preferences">
136      <?gmfgen generated="true"?>
137      <initializer class="org.jat.exhibition.diagram.preferences.
           DiagramPreferenceInitializer"/>
138   </extension>
139
140   <extension point="org.eclipse.ui.preferencePages">
141      <?gmfgen generated="true"?>
142      <page
143            id="JAT.diagram.general"
144            name="%preference.page.title.JAT.diagram.general"
145            class="org.jat.exhibition.diagram.preferences.
                   DiagramGeneralPreferencePage">
146      </page>
147      <page
148            id="JAT.diagram.appearance"
149            name="%preference.page.title.JAT.diagram.appearance"
150            category="JAT.diagram.general"
151            class="org.jat.exhibition.diagram.preferences.
                   DiagramAppearancePreferencePage">
152      </page>
153      <page
154            id="JAT.diagram.connections"
155            name="%preference.page.title.JAT.diagram.connections"
156            category="JAT.diagram.general"
157            class="org.jat.exhibition.diagram.preferences.
                   DiagramConnectionsPreferencePage">
158      </page>
159      <page
160            id="JAT.diagram.printing"
161            name="%preference.page.title.JAT.diagram.printing"
162            category="JAT.diagram.general"
163            class="org.jat.exhibition.diagram.preferences.
                   DiagramPrintingPreferencePage">
164      </page>
165      <page
166            id="JAT.diagram.rulersAndGrid"
167            name="%preference.page.title.JAT.diagram.rulersAndGrid"
168            category="JAT.diagram.general"
```

```
169             class="org.jat.exhibition.diagram.preferences.
                    DiagramRulersAndGridPreferencePage">
170       </page>
171       <page
172             id="JAT.diagram.pathmaps"
173             name="%preference.page.title.JAT.diagram.pathmaps"
174             category="JAT.diagram.general"
175             class="org.eclipse.gmf.runtime.emf.ui.preferences.
                    PathmapsPreferencePage">
176       </page>
177    </extension>
178
179
180    <extension point="org.eclipse.ui.views.properties.tabbed.
           propertyContributor">
181       <?gmfgen generated="true"?>
182       <propertyContributor
183             contributorId="org.jat.exhibition.diagram"
184             labelProvider="org.jat.exhibition.diagram.sheet.
                    ExhibitionSheetLabelProvider">
185          <propertyCategory category="domain"/>
186          <propertyCategory category="visual"/>
187          <propertyCategory category="extra"/>
188       </propertyContributor>
189    </extension>
190
191    <extension point="org.eclipse.ui.views.properties.tabbed.propertyTabs">
192       <?gmfgen generated="true"?>
193       <propertyTabs contributorId="org.jat.exhibition.diagram">
194          <propertyTab
195              category="visual"
196              id="property.tab.AppearancePropertySection"
197              label="%tab.appearance"/>
198           <propertyTab
199              category="visual"
200              id="property.tab.DiagramPropertySection"
201              label="%tab.diagram"/>
202           <propertyTab
203              category="domain"
204              id="property.tab.domain"
205              label="%tab.domain"/>
206       </propertyTabs>
207    </extension>
208
209    <extension point="org.eclipse.ui.views.properties.tabbed.
           propertySections">
210       <?gmfgen generated="false"?>
211       <propertySections contributorId="org.jat.exhibition.diagram">
212          <propertySection id="property.section.
                ConnectorAppearancePropertySection"
213             filter="org.eclipse.gmf.runtime.diagram.ui.properties.filters.
```

```
                ConnectionEditPartPropertySectionFilter"
214             class="org.eclipse.gmf.runtime.diagram.ui.properties.sections.
                    appearance.ConnectionAppearancePropertySection"
215             tab="property.tab.AppearancePropertySection">
216          </propertySection>
217          <propertySection id="property.section.
                 ShapeColorAndFontPropertySection"
218             filter="org.eclipse.gmf.runtime.diagram.ui.properties.filters.
                    ShapeEditPartPropertySectionFilter"
219             class="org.eclipse.gmf.runtime.diagram.ui.properties.sections.
                    appearance.ShapeColorsAndFontsPropertySection"
220             tab="property.tab.AppearancePropertySection">
221          </propertySection>
222          <propertySection id="property.section.
                 DiagramColorsAndFontsPropertySection"
223             filter="org.eclipse.gmf.runtime.diagram.ui.properties.filters.
                    DiagramEditPartPropertySectionFilter"
224             class="org.eclipse.gmf.runtime.diagram.ui.properties.sections.
                    appearance.DiagramColorsAndFontsPropertySection"
225             tab="property.tab.AppearancePropertySection">
226          </propertySection>
227          <propertySection id="property.section.RulerGridPropertySection"
228             filter="org.eclipse.gmf.runtime.diagram.ui.properties.filters.
                    DiagramEditPartPropertySectionFilter"
229             class="org.eclipse.gmf.runtime.diagram.ui.properties.sections.
                    grid.RulerGridPropertySection"
230             tab="property.tab.DiagramPropertySection">
231          </propertySection>
232          <propertySection
233             id="property.section.domain"
234             tab="property.tab.domain"
235             class="org.jat.exhibition.diagram.sheet.
                    ExhibitionPropertySection">
236             <input type="org.eclipse.gmf.runtime.notation.View"/>
237             <input type="org.eclipse.gef.EditPart"/>
238             <input type="org.eclipse.emf.ecore.EObject"/>
239          </propertySection>
240       </propertySections>
241    </extension>
242
243    <extension point="org.eclipse.gmf.runtime.diagram.core.viewProviders">
244       <?gmfgen generated="true"?>
245       <viewProvider class="org.jat.exhibition.diagram.providers.
              ExhibitionViewProvider">
246          <Priority name="Lowest"/>
247          <context viewClass="org.eclipse.gmf.runtime.notation.Diagram"
                 semanticHints="Exhibition"/>
248          <context viewClass="org.eclipse.gmf.runtime.notation.Node"
                 semanticHints=""/>
249          <context viewClass="org.eclipse.gmf.runtime.notation.Edge"
                 semanticHints=""/>
```

```
250       </viewProvider>
251    </extension>
252
253    <extension point="org.eclipse.gmf.runtime.diagram.ui.editpartProviders">
254       <?gmfgen generated="true"?>
255       <editpartProvider class="org.jat.exhibition.diagram.providers.
              ExhibitionEditPartProvider">
256          <Priority name="Lowest"/>
257       </editpartProvider>
258    </extension>
259
260    <extension point="org.eclipse.gmf.runtime.emf.ui.
           modelingAssistantProviders">
261       <?gmfgen generated="true"?>
262       <modelingAssistantProvider class="org.jat.exhibition.diagram.
              providers.ExhibitionModelingAssistantProvider">
263          <Priority name="Lowest"/>
264       </modelingAssistantProvider>
265    </extension>
266
267    <extension point="org.eclipse.gmf.runtime.common.ui.services.
           iconProviders">
268       <?gmfgen generated="true"?>
269       <IconProvider class="org.jat.exhibition.diagram.providers.
              ExhibitionIconProvider">
270          <Priority name="Low"/>
271       </IconProvider>
272    </extension>
273
274    <extension point="org.eclipse.gmf.runtime.common.ui.services.
           parserProviders">
275       <?gmfgen generated="true"?>
276       <ParserProvider class="org.jat.exhibition.diagram.providers.
              ExhibitionParserProvider">
277          <Priority name="Lowest"/>
278       </ParserProvider>
279    </extension>
280
281    <extension point="org.eclipse.gmf.runtime.emf.type.core.elementTypes">
282       <?gmfgen generated="true"?>
283       <metamodel nsURI="http:///org/jat/exhibition.ecore">
284          <metamodelType
285                id="org.jat.exhibition.diagram.Exhibition_1000"
286                name="%metatype.name.Exhibition_1000"
287                kind="org.eclipse.gmf.runtime.emf.type.core.IHintedType"
288                eclass="Exhibition"
289                edithelper="org.jat.exhibition.diagram.edit.helpers.
                       ExhibitionEditHelper">
290             <param name="semanticHint" value="1000"/>
291          </metamodelType>
292       </metamodel>
```

```
293       <metamodel nsURI="http:///org/jat/exhibition.ecore">
294          <metamodelType
295                id="org.jat.exhibition.diagram.Scene_2002"
296                name="%metatype.name.Scene_2002"
297                kind="org.eclipse.gmf.runtime.emf.type.core.IHintedType"
298                eclass="Scene"
299                edithelper="org.jat.exhibition.diagram.edit.helpers.
                       SceneEditHelper">
300             <param name="semanticHint" value="2002"/>
301          </metamodelType>
302       </metamodel>
303       <metamodel nsURI="http:///org/jat/exhibition.ecore">
304          <metamodelType
305                id="org.jat.exhibition.diagram.ExhibitionModule_2003"
306                name="%metatype.name.ExhibitionModule_2003"
307                kind="org.eclipse.gmf.runtime.emf.type.core.IHintedType"
308                eclass="ExhibitionModule"
309                edithelper="org.jat.exhibition.diagram.edit.helpers.
                       ExhibitionModuleEditHelper">
310             <param name="semanticHint" value="2003"/>
311          </metamodelType>
312       </metamodel>
313       <metamodel nsURI="http:///org/jat/exhibition.ecore">
314          <metamodelType
315                id="org.jat.exhibition.diagram.SceneLink_3003"
316                name="%metatype.name.SceneLink_3003"
317                kind="org.eclipse.gmf.runtime.emf.type.core.IHintedType"
318                eclass="SceneLink"
319                edithelper="org.jat.exhibition.diagram.edit.helpers.
                       SceneLinkEditHelper">
320             <param name="semanticHint" value="3003"/>
321          </metamodelType>
322       </metamodel>
323       <metamodel nsURI="http:///org/jat/exhibition.ecore">
324          <metamodelType
325                id="org.jat.exhibition.diagram.ExhibitionModuleLink_3004"
326                name="%metatype.name.ExhibitionModuleLink_3004"
327                kind="org.eclipse.gmf.runtime.emf.type.core.IHintedType"
328                eclass="ExhibitionModuleLink"
329                edithelper="org.jat.exhibition.diagram.edit.helpers.
                       ExhibitionModuleLinkEditHelper">
330             <param name="semanticHint" value="3004"/>
331          </metamodelType>
332       </metamodel>
333       <metamodel nsURI="http:///org/jat/exhibition.ecore">
334          <specializationType
335                id="org.jat.exhibition.diagram.
                       ExhibitionModuleLinkExhibitionModuleTarget_4005"
336                name="%metatype.name.
                       ExhibitionModuleLinkExhibitionModuleTarget_4005"
337                kind="org.eclipse.gmf.runtime.emf.type.core.IHintedType">
```

```
338             <specializes id="org.eclipse.gmf.runtime.emf.type.core.null"/>
339             <param name="semanticHint" value="4005"/>
340          </specializationType>
341       </metamodel>
342       <metamodel nsURI="http:///org/jat/exhibition.ecore">
343          <specializationType
344                id="org.jat.exhibition.diagram.SceneLinkSceneLinkTarget_4006
                   "
345                name="%metatype.name.SceneLinkSceneLinkTarget_4006"
346                kind="org.eclipse.gmf.runtime.emf.type.core.IHintedType">
347             <specializes id="org.eclipse.gmf.runtime.emf.type.core.null"/>
348             <param name="semanticHint" value="4006"/>
349          </specializationType>
350       </metamodel>
351    </extension>
352
353    <extension point="org.eclipse.gmf.runtime.emf.type.core.
          elementTypeBindings">
354       <?gmfgen generated="true"?>
355       <clientContext id="ExhibitionClientContext">
356          <enablement>
357             <test
358                property="org.eclipse.gmf.runtime.emf.core.editingDomain"
359                value="org.jat.exhibition.diagram.EditingDomain"/>
360          </enablement>
361       </clientContext>
362       <binding context="ExhibitionClientContext">
363          <elementType ref="org.jat.exhibition.diagram.Exhibition_1000"/>
364          <elementType ref="org.jat.exhibition.diagram.Scene_2002"/>
365          <elementType ref="org.jat.exhibition.diagram.ExhibitionModule_2003
                "/>
366          <elementType ref="org.jat.exhibition.diagram.SceneLink_3003"/>
367          <elementType ref="org.jat.exhibition.diagram.
                ExhibitionModuleLink_3004"/>
368          <elementType ref="org.jat.exhibition.diagram.
                ExhibitionModuleLinkExhibitionModuleTarget_4005"/>
369          <elementType ref="org.jat.exhibition.diagram.
                SceneLinkSceneLinkTarget_4006"/>
370          <advice ref="org.eclipse.gmf.runtime.diagram.core.advice.
                notationDepdendents"/>
371       </binding>
372    </extension>
373
374    <extension id="ExhibitionApplication" point="org.eclipse.core.runtime.
          applications">
375       <?gmfgen generated="true"?>
376       <application>
377          <run class="org.jat.exhibition.diagram.application.
                ExhibitionApplication"/>
378       </application>
379    </extension>
```

```
380
381    <extension point="org.eclipse.ui.perspectives">
382       <?gmfgen generated="true"?>
383       <perspective
384             id="org.jat.exhibition.diagram.ExhibitionPerspective"
385             name="%perspectiveName"
386             class="org.jat.exhibition.diagram.application.
                    DiagramEditorPerspective">
387       </perspective>
388    </extension>
389
390    <extension point="org.eclipse.ui.commands">
391       <?gmfgen generated="false"?>
392       <!--<command
393             name="%openURIActionLabel"
394             description="%openURIActionDescription"
395             categoryId="org.eclipse.ui.category.file"
396             id="org.jat.exhibition.diagram.OpenURICommand"/> -->
397       <command
398             name="%openActionLabel"
399             description="%openActionDescription"
400             categoryId="org.eclipse.ui.category.file"
401             id="org.jat.exhibition.diagram.OpenCommand"/>
402    </extension>
403
404    <extension point="org.eclipse.ui.bindings">
405       <?gmfgen generated="false"?>
406       <!--<key
407             commandId="org.jat.exhibition.diagram.OpenURICommand"
408             sequence="M1+U"
409             schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>-->
410       <key
411             commandId="org.jat.exhibition.diagram.OpenCommand"
412             sequence="M1+O"
413             schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>
414    </extension>
415
416    <extension point="org.eclipse.ui.editorActions">
417          <editorContribution
418          id="org.eclipse.ui.articles.action.contribution.editor2"
419             targetID="org.eclipse.ui.DefaultTextEditor">
420             <action
421                id="org.eclipse.ui.articles.action.contribution.editor.
                       action1"
422                label="Generate HTML"
423                tooltip="Generates HTML pages for this exhibition."
424          class="org.jat.exhibition.diagram.application.actions.
                    TransformToHtmlAction">
425       </action>
426          </editorContribution>
427    </extension>
```

```
428
429    <extension point="org.eclipse.ui.actionSets">
430       <?gmfgen generated="false"?>
431       <actionSet
432             label="%applicationActionSetLabel"
433             visible="true"
434             id="org.jat.exhibition.diagram.ActionSet">
435          <action
436                label="%newDiagramActionLabel"
437                class="org.jat.exhibition.diagram.application.
                       DiagramEditorActionBarAdvisor$NewDiagramAction"
438                menubarPath="file/new/additions"
439                id="org.jat.exhibition.diagram.NewDiagramAction">
440          </action>
441          <action
442                label="%aboutActionLabel"
443                class="org.jat.exhibition.diagram.application.
                       DiagramEditorActionBarAdvisor$AboutAction"
444                menubarPath="help/additions"
445                id="org.jat.exhibition.diagram.AboutAction">
446          </action>
447         <!-- <action
448                label="%openURIActionLabel"
449                definitionId="org.jat.exhibition.diagram.OpenURICommand"
450                class="org.jat.exhibition.diagram.application.
                       DiagramEditorActionBarAdvisor$OpenURIAction"
451                menubarPath="file/additions"
452                id="org.jat.exhibition.diagram.OpenURIAction">
453          </action>-->
454          <action
455                label="%openActionLabel"
456                definitionId="org.jat.exhibition.diagram.OpenCommand"
457                class="org.jat.exhibition.diagram.application.
                       DiagramEditorActionBarAdvisor$OpenAction"
458                menubarPath="file/additions"
459                id="org.jat.exhibition.diagram.OpenAction">
460          </action>
461       </actionSet>
462    </extension>
463    <extension point="org.eclipse.ui.commands">
464       <?gmfgen generated="true"?>
465       <command
466             name="%openURIActionLabel"
467             description="%openURIActionDescription"
468             categoryId="org.eclipse.ui.category.file"
469             id="org.jat.exhibition.diagram.OpenURICommand"/>
470       <command
471             name="%openActionLabel"
472             description="%openActionDescription"
473             categoryId="org.eclipse.ui.category.file"
474             id="org.jat.exhibition.diagram.OpenCommand"/>
```

```
475    </extension>
476
477    <extension point="org.eclipse.ui.bindings">
478       <?gmfgen generated="true"?>
479       <key
480             commandId="org.jat.exhibition.diagram.OpenURICommand"
481             sequence="M1+U"
482             schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>
483       <key
484             commandId="org.jat.exhibition.diagram.OpenCommand"
485             sequence="M1+O"
486             schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"/>
487    </extension>
488
489    <extension
490          point="org.eclipse.ui.views">
491       <view
492             name="View"
493             allowMultiple="false"
494             class="org.jat.exhibition.diagram.rcpviews.modelnavigator.
                  ModelNavigatorView"
495             id="org.jat.exhibition.NavigatorView">
496       </view>
497       <view
498             name="Validation Results"
499             class="org.jat.exhibition.diagram.rcpviews.validation.
                  ValidationResultView"
500             id="org.jat.exhibition.ValidationResultView">
501       </view>
502    </extension>
503    <extension
504          id="product"
505          point="org.eclipse.core.runtime.products">
506       <product
507             application="org.jat.exhibition.diagram.ExhibitionApplication"
508             name="JAT">
509          <property
510                name="appName"
511                value="JAT">
512          </property>
513       </product>
514    </extension><extension id="ValidationContributionItemProvider" name="
          Validation"
515       point="org.eclipse.gmf.runtime.common.ui.services.action.
             contributionItemProviders">
516       <?gmfgen generated="false"?>
517       <contributionItemProvider
518          class="org.jat.exhibition.diagram.providers.
                ExhibitionValidationProvider">
519          <Priority name="Low"/>
520          <partContribution class="org.eclipse.gmf.runtime.diagram.ui.
```

```
            resources.editor.parts.DiagramDocumentEditor">
521         <partMenuGroup menubarPath="/diagramMenu/" id="validationGroup"
                />
522         <partAction id="validateAction" menubarPath="/diagramMenu/
                validationGroup"/>
523       </partContribution>
524     </contributionItemProvider>
525   </extension>
526   <extension id="validationDecoratorProvider" name="ValidationDecorations"
        point="org.eclipse.gmf.runtime.diagram.ui.decoratorProviders">
527     <?gmfgen generated="true"?>
528     <decoratorProvider class="org.jat.exhibition.diagram.providers.
          ExhibitionValidationDecoratorProvider">
529       <Priority name="Lowest"/>
530       <object class="org.eclipse.gmf.runtime.diagram.ui.editparts.
            IPrimaryEditPart(org.eclipse.gmf.runtime.diagram.ui)" id="
            PRIMARY_VIEW"/>
531       <context decoratorTargets="PRIMARY_VIEW"/>
532     </decoratorProvider>
533   </extension>
534   <extension point="org.eclipse.emf.validation.constraintProviders">
535     <?gmfgen generated="true"?>
536         <category id="exhibitionAuditContainer" mandatory="false" name=
              "Exhibition Audit Container">
537       <![CDATA[]]>
538     </category>
539     <category id="exhibitionAuditContainer/sceneAuditContainer" mandatory
          ="false" name="Scene Audit Container">
540       <![CDATA[Audit Container for Scenes.]]>
541     </category>
542     <category id="exhibitionAuditContainer/linkAuditContainer" mandatory=
          "false" name="Link Audit Container">
543       <![CDATA[]]>
544     </category>
545     <category id="exhibitionAuditContainer/exhibitionModuleAuditContainer
          " mandatory="false" name="Exhibition Module Audit Container">
546       <![CDATA[]]>
547     </category>
548     <constraintProvider cache="true">
549       <package namespaceUri="http:///org/jat/exhibition.ecore"/>
550       <package namespaceUri="http:///org/jat/exhibition.ecore"/>
551       <package namespaceUri="http:///org/jat/exhibition.ecore"/>
552       <package namespaceUri="http:///org/jat/exhibition.ecore"/>
553       <package namespaceUri="http:///org/jat/exhibition.ecore"/>
554           <constraints categories="exhibitionAuditContainer">
555             <constraint id="exhibitionStartSceneRule"
556           lang="Java" class="org.jat.exhibition.diagram.providers.
                ExhibitionValidationProvider$Adapter1"
557           name="No Start Scene chosen."
558           mode="Batch"
559           severity="WARNING" statusCode="200">
```

```
560
561             <description><![CDATA[]]></description>
562             <message><![CDATA[There is no Start Scene chosen for the
                    Exhibition.]]></message>
563             <target class="exhibition.Exhibition"/>
564           </constraint>
565                   </constraints>
566           <constraints categories="exhibitionAuditContainer/
                sceneAuditContainer">
567                   <constraint id="noSceneBackgroundImage"
568             lang="Java" class="org.jat.exhibition.diagram.providers.
                    ExhibitionValidationProvider$Adapter2"
569             name="No background image selected."
570             mode="Batch"
571             severity="WARNING" statusCode="200">
572
573             <description><![CDATA[]]></description>
574             <message><![CDATA[There is no background image selected for the
                     Scene.]]></message>
575             <target class="exhibition.Scene"/>
576           </constraint>
577                   <constraint id="sceneNotReferenced"
578             lang="Java" class="org.jat.exhibition.diagram.providers.
                    ExhibitionValidationProvider$Adapter3"
579             name="Scene is not referenced."
580             mode="Batch"
581             severity="WARNING" statusCode="200">
582
583             <description><![CDATA[]]></description>
584             <message><![CDATA[The Scene is not referenced from any other
                    Scene. It will not be accessible in the Exhibition.]]></
                    message>
585             <target class="exhibition.Scene"/>
586           </constraint>
587                   </constraints>
588           <constraints categories="exhibitionAuditContainer/
                linkAuditContainer">
589                   <constraint id="notarget"
590             lang="Java" class="org.jat.exhibition.diagram.providers.
                    ExhibitionValidationProvider$Adapter4"
591             name="No target selected."
592             mode="Batch"
593             severity="ERROR" statusCode="200">
594
595             <description><![CDATA[]]></description>
596             <message><![CDATA[There is no target selected for the Link.]]><
                    /message>
597             <target class="exhibition.Link"/>
598           </constraint>
599                   </constraints>
600           <constraints categories="exhibitionAuditContainer/
```

```
            exhibitionModuleAuditContainer">
601                 <constraint id="exhibitionModuleIsLinked"
602           lang="Java" class="org.jat.exhibition.diagram.providers.
                  ExhibitionValidationProvider$Adapter5"
603           name="Exhibition Module not referenced."
604           mode="Batch"
605           severity="WARNING" statusCode="200">
606
607           <description><![CDATA[]]></description>
608           <message><![CDATA[The Exhibition Module is not referenced from
                  any Scene. It will not be accessible in the Exhibition.]]><
                  /message>
609           <target class="exhibition.ExhibitionModule"/>
610        </constraint>
611                 </constraints>
612     </constraintProvider>
613  </extension>
614  <extension point="org.eclipse.emf.validation.constraintBindings">
615     <?gmfgen generated="true"?>
616                 <clientContext default="false" id="org.jat.exhibition.
                        diagram.DefaultCtx">
617        <selector class="org.jat.exhibition.diagram.providers.
              ExhibitionValidationProvider$DefaultCtx1"/>
618     </clientContext>
619     <binding context="org.jat.exhibition.diagram.DefaultCtx">
620                 <constraint ref="org.jat.exhibition.diagram.
                        exhibitionStartSceneRule"/>
621                 <constraint ref="org.jat.exhibition.diagram.
                        noSceneBackgroundImage"/>
622                 <constraint ref="org.jat.exhibition.diagram.
                        sceneNotReferenced"/>
623                 <constraint ref="org.jat.exhibition.diagram.notarget"/>
624                 <constraint ref="org.jat.exhibition.diagram.
                        exhibitionModuleIsLinked"/>
625              </binding>
626              </extension>
627  <extension point="org.eclipse.ui.commands">
628     <?gmfgen generated="true"?>
629     <command
630        categoryId="org.eclipse.ui.category.edit"
631        defaultHandler="org.jat.exhibition.diagram.part.
              ExhibitionDiagramUpdateCommand"
632        description="%update.diagram.description"
633        id="org.jat.exhibition.diagram.updateDiagram"
634        name="%update.diagram.name"/>
635  </extension>
636  <extension point="org.eclipse.ui.bindings">
637     <?gmfgen generated="true"?>
638     <key
639        commandId="org.jat.exhibition.diagram.updateDiagram"
640        contextId="org.jat.exhibition.diagram.ui.diagramContext"
```

```
641          schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"
642          sequence="F5"/>
643    </extension>
644 </plugin>
```

Listing D-8 Ausschnitt der Ecore-Datei zur Definition des Metamodells

D.6.2. Quelltext der Klasse `ValidationMarker`

Listing D-9 zeigt den Quelltext der Klasse `ValidationMarker`, der ein zweiter Konstruktor hinzugefügt wurde. Diesem Konstruktor wird zusätzlich das validierte Modellelement übergeben.

```
1  package org.jat.exhibition.diagram.part;
2
3  import java.util.Collections;
4  import java.util.HashMap;
5  import java.util.HashSet;
6  import java.util.Map;
7  import java.util.Set;
8
9  import org.eclipse.emf.ecore.EObject;
10 import org.eclipse.gef.EditPartViewer;
11
12 /**
13  * @generated
14  */
15 public class ValidationMarker {
16
17    /**
18     * @generated NOT
19     */
20    public static final String KEY = "validation_marker"; //$NON-NLS-1$
21
22    /**
23     * @generated
24     */
25    public static final ValidationMarker[] EMPTY_ARRAY = new
          ValidationMarker[0];
26
27    /**
28     * @generated
29     */
30    private final String location;
31
32    /**
33     * @generated
34     */
35    private final String message;
```

```
36
37    private final EObject element;
38
39    /**
40     * @generated
41     */
42    private final int statusSeverity;
43
44    /**
45     * @generated NOT
46     */
47    public ValidationMarker(String location, String message, int
          statusSeverity) {
48       this.location = location;
49       this.message = message;
50       this.statusSeverity = statusSeverity;
51       this.element = null;
52    }
53
54    /**
55     * @generated NOT
56     */
57    public ValidationMarker(String location, String message,
58          int statusSeverity, EObject element) {
59       System.out.println(getClass().getName() + " add marker " + location);
60       this.location = location;
61       this.message = message;
62       this.statusSeverity = statusSeverity;
63       this.element = element;
64    }
65
66    /**
67     * @generated
68     */
69    public String getLocation() {
70       return location;
71    }
72
73    /**
74     * @generated
75     */
76    public String getMessage() {
77       return message;
78    }
79
80    /**
81     * @generated
82     */
83    public int getStatusSeverity() {
84       return statusSeverity;
85    }
```

```
86
87      /**
88       * @generated
89       */
90      private static Map getMarkers(EditPartViewer viewer) {
91         Map markers = (Map) viewer.getProperty(KEY);
92         if (markers == null) {
93            markers = new HashMap();
94            viewer.setProperty(KEY, markers);
95         }
96         return markers;
97      }
98
99      /**
100      * @generated
101      */
102     private static Set getMarkers(EditPartViewer viewer, String viewId,
103           boolean create) {
104        Set markers = (Set) getMarkers(viewer).get(viewId);
105        if (markers == null) {
106           if (!create) {
107              return Collections.EMPTY_SET;
108           }
109           markers = new HashSet();
110           getMarkers(viewer).put(viewId, markers);
111        }
112        return markers;
113     }
114
115     /**
116      * @generated
117      */
118     public static ValidationMarker[] getMarkers(EditPartViewer viewer,
119           String viewId) {
120        Set markers = getMarkers(viewer, viewId, false);
121        if (markers.isEmpty()) {
122           return EMPTY_ARRAY;
123        }
124        return (ValidationMarker[]) markers
125              .toArray(new ValidationMarker[markers.size()]);
126     }
127
128     /**
129      * @generated
130      */
131     public static void removeAllMarkers(EditPartViewer viewer) {
132        getMarkers(viewer).clear();
133     }
134
135     /**
136      * @generated
```

```
137      */
138     public void add(EditPartViewer viewer, String viewId) {
139        getMarkers(viewer, viewId, true).add(this);
140     }
141
142     public EObject getElement() {
143        return element;
144     }
145  }
```

Listing D-9 Quelltext der Klasse `ValidationMarker`

D.6.3. Quelltext des Label Providers der Tabelle der Validierungsergebnisse

Listing D-10 zeigt den Quelltext des Label Providers für die Tabelle der Validierungsergebnisse.

```
1  package org.jat.exhibition.diagram.rcpviews.validation;
2
3  import java.net.URL;
4
5  import org.eclipse.emf.ecore.EObject;
6  import org.eclipse.emf.edit.provider.DelegatingWrapperItemProvider;
7  import org.eclipse.emf.validation.model.IConstraintStatus;
8  import org.eclipse.jface.resource.ImageDescriptor;
9  import org.eclipse.jface.viewers.ILabelProviderListener;
10 import org.eclipse.jface.viewers.ITableLabelProvider;
11 import org.eclipse.swt.graphics.Image;
12 import org.eclipse.swt.graphics.ImageData;
13 import org.eclipse.ui.PlatformUI;
14 import org.jat.exhibition.ExhibitionPackage;
15 import org.jat.exhibition.diagram.part.ExhibitionDiagramEditorPlugin;
16 import org.jat.exhibition.diagram.part.ValidationMarker;
17 import org.jat.exhibition.diagram.util.ExhibitionJatPath;
18 import org.jat.exhibition.provider.ExhibitionEditPlugin;
19 import org.jat.filehandler.services.FileHandler;
20
21 public class ValidationResultLabelProvider implements ITableLabelProvider {
22
23    public Image getColumnImage(Object element, int columnIndex) {
24       //System.out.println(getClass().getName() + " get image " + element);
25       if (element instanceof ValidationResult) {
26          switch (columnIndex) {
27          case 0:
28          {
29
30             ValidationMarker markerElement = ((ValidationResult) element).
                   marker;
```

```
31              ImageDescriptor imageDesc = null;
32              switch(markerElement.getStatusSeverity())
33              {
34              case IConstraintStatus.ERROR :
35                  imageDesc = ExhibitionDiagramEditorPlugin.
                        getBundledImageDescriptor(ExhibitionJatPath.ERROR_IMG_16
                        );
36                  break;
37              case IConstraintStatus.WARNING :
38                  imageDesc = ExhibitionDiagramEditorPlugin.
                        getBundledImageDescriptor(ExhibitionJatPath.
                        WARNING_IMG_16);
39                  break;
40              default :
41                  imageDesc = ExhibitionDiagramEditorPlugin.
                        getBundledImageDescriptor(ExhibitionJatPath.INFO_IMG_16)
                        ;
42              }
43              return imageDesc.createImage();
44          }
45          case 1: {
46              Object markerElement = ((ValidationResult) element).marker.
                    getElement();
47              if (markerElement != null)
48              {
49                  DelegatingWrapperItemProvider delWrapItemProvider = new
                        DelegatingWrapperItemProvider((EObject)markerElement, (
                        EObject)markerElement, ExhibitionPackage.eINSTANCE.
                        eContainingFeature(), 0, ExhibitionDiagramEditorPlugin.
                        getInstance().getItemProvidersAdapterFactory());
50                  URL url = (URL) delWrapItemProvider.getImage(markerElement);
51                  String imagePath = url.getPath();
52                  //System.out.println(getClass().getName() + " plugin " +
                        ExhibitionEditPlugin.getPlugin().getSymbolicName());
53                  String absolutePath = FileHandler.getAbsolutePath(
                        ExhibitionEditPlugin.getPlugin().getSymbolicName(),
                        imagePath);
54                  ImageData data = new ImageData(absolutePath);
55                  return new Image(PlatformUI.getWorkbench().getDisplay(),
                        data);
56              }
57              return null;
58          }
59          case 2: {
60              return null;
61          }
62          }
63      }
64      return null;
65  }
66
```

```
67     public String getColumnText(Object element, int columnIndex) {
68        //System.out.println(getClass().getName() + " get text " + element);
69        if (element instanceof ValidationResult) {
70           switch (columnIndex) {
71           case 1: {
72              EObject markerElement = ((ValidationResult) element).marker.
                   getElement();
73              if (markerElement != null)
74              {
75                 DelegatingWrapperItemProvider delWrapItemProvider = new
                       DelegatingWrapperItemProvider(markerElement,
                       markerElement, ExhibitionPackage.eINSTANCE.
                       eContainingFeature(), 0, ExhibitionDiagramEditorPlugin.
                       getInstance().getItemProvidersAdapterFactory());
76                 return delWrapItemProvider.getText(markerElement);
77              }
78              return "";
79           }
80           case 2: {
81              return ((ValidationResult) element).marker.getMessage();
82           }
83           }
84        }
85
86        return "";
87     }
88
89     public void addListener(ILabelProviderListener listener) {
90        // TODO Auto-generated method stub
91
92     }
93
94     public void dispose() {
95        // TODO Auto-generated method stub
96
97     }
98
99     public boolean isLabelProperty(Object element, String property) {
100       // TODO Auto-generated method stub
101       return false;
102    }
103
104    public void removeListener(ILabelProviderListener listener) {
105       // TODO Auto-generated method stub
106
107    }
108
109 }
```

Listing D-10 Quelltext des Label Providers der Tabelle der Validierungsergebnisse

D.7. Unit-Tests

Abbildung D-17 zeigt die Ergebnisse der JUnit-Tests, die unter Verwendung von Eclipse durchgeführt wurden, für das Plugin `org.jat.generation`.

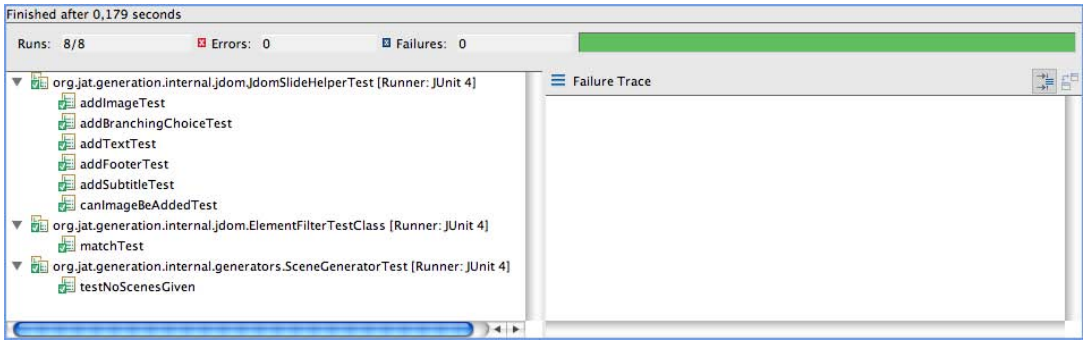


Abb. D-17. JUnit-Tests des Plugins `org.jat.generation`

Abbildung D-18 zeigt die Ergebnisse der JUnit-Tests, die unter Verwendung von Eclipse durchgeführt wurden, für das Plugin `org.jat.generation.engine`.

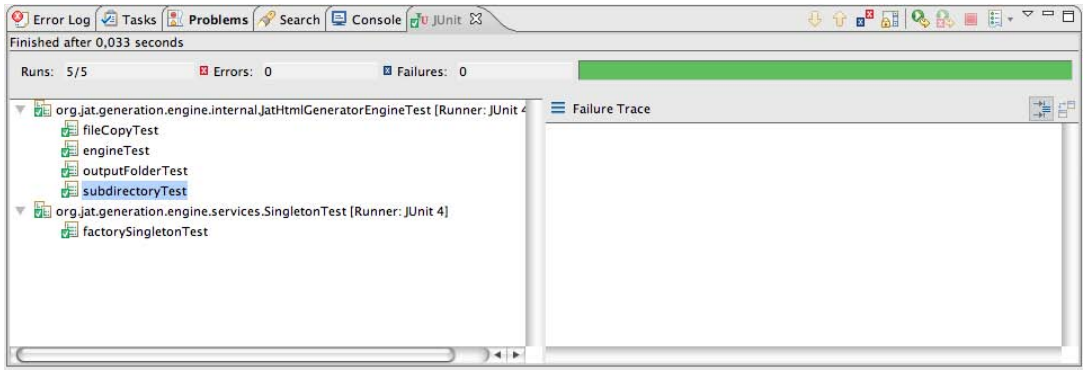


Abb. D-18. JUnit-Tests des Plugins `org.jat.generation.engine`

Anhang E.

Ergänzungen zu den Anwendertests

E.1. Anwendertest

Nachfolgend der Testbogen des Anwendertests.

Anwendertest – JAT

Sie werden eine kurze Einführung in die Bedienung von JAT erhalten. Im Anschluss versuchen Sie bitte, die nachfolgenden Aufgaben zu lösen. Notieren Sie vor Beginn und nach Beendigung jeder Aufgabe die Uhrzeit (bis auf eine halbe Minute genau), um eine Analyse des Zeitaufwandes zu ermöglichen. Neben jeder Aufgabe finden Sie Ankreuzkästchen, mit Hilfe derer Sie den Schwierigkeitsgrad der Aufgabe bewerten können. Können Sie eine Aufgabe nicht lösen, kreuzen Sie „nicht lösbar“ an. Falls Sie für die Lösung einer Aufgabe weitere Hilfe beanspruchen, kreuzen Sie bitte „zusätzliche Hilfe war notwendig“ zusätzlich zu Ihrer Einschätzung des Schwierigkeitsgrades an.

Bevor Sie mit den Aufgaben beginnen, suchen Sie sich bitte einige Bilder und Texte für die Verwendung in der Virtuellen Ausstellung heraus. Vergessen Sie nicht das Projekt nach jeder Aufgabe zu speichern.

Schwierigkeitsgrad (1= einfach, 4=schwer)

Aufgabe 1:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Legen Sie eine neue *virtuelle Ausstellung* an.

Aufgabe 2:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Legen Sie zwei *Scenes* an. Geben Sie jeder *Scene* ein Hintergrundbild und einen Titel.

Aufgabe 3:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Erstellen Sie zwei *Scene Links*. Geben Sie beiden *Scene Links* einen Titel. Verbinden Sie die erste *Scene* mit der zweiten durch einen der beiden zuvor erstellen *Scene Links*.

Aufgabe 4:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Erstellen Sie ein *Exhibition Module* und geben Sie ihm einen Titel.

Aufgabe 5:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Öffnen Sie das Unterdiagram für das zuvor erstellte *Exhibition Module*. Legen Sie einen Slide an. Fügen Sie zu dem *Slide* zwei Bilder und einen Text von den zu Anfang bereitgelegten Bildern und Texten hinzu. Geben Sie dem *Slide* einen Titel, Untertitel und Fußtext. Weisen Sie dem *Slide* ein Template zu, welches zwei Bilder und einen Text anzeigt.

Aufgabe 6:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Erstellen Sie zwei weitere *Slides* ähnlich dem vorangegangenen, aber fügen Sie jeweils nur ein Bild hinzu. Ändern Sie bei einem Slide, die Größe des in der Vorschau gezeigten Bildes auf 600x600.

Aufgabe 7:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Erstellen Sie einen *Branching Point*. Fügen Sie dem *Branching Point* einen Text und ein Bild hinzu und geben Sie ihm einen Titel, Untertitel und einen Fußtext. Setzen Sie die „Number of columns of choices“ auf 2.

Aufgabe 8:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Erstellen Sie eine *Sequence* und geben Sie ihr einen Titel. Öffnen Sie den Slideauswahl-Dialog für diese *Sequence* und fügen Sie der *Sequence* die beiden zuerst erstellen *Slides* und den *Branching Point* hinzu. Schließen Sie den Dialog.

Aufgabe 9:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar
☐ zusätzliche Hilfe war notwendig

Beginn: Uhr
Ende: Uhr

Erstellen Sie eine zweite *Sequence* und geben Sie ihr einen Titel. Fügen Sie der *Sequence* den dritten bisher in keiner *Sequence* verwendeten *Slide* hinzu.

Aufgabe 10: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar

Beginn: Uhr ☐ zusätzliche Hilfe war notwendig

Ende: Uhr

Fügen Sie dem zuvor erstellten *Branching Point* zwei *Branching Point Choices* hinzu. Wählen Sie als Ziel-*Sequence* der ersten *Branching Point Choice* die *Sequence*, die Sie als zweites angelegt haben.

Aufgabe 11: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar

Beginn: Uhr ☐ zusätzliche Hilfe war notwendig

Ende: Uhr

Validieren Sie das Modell und öffnen Sie die Tabelle mit den Ergebnissen der Validierung. Bearbeiten Sie die Fehler und Warnungen wie folgt.

Aufgabe 12: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar

Beginn: Uhr ☐ zusätzliche Hilfe war notwendig

Ende: Uhr

Wählen Sie als *Startsequence* des *Exhibition Modules* die als erstes erstellte *Sequence*.

Aufgabe 13: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar

Beginn: Uhr ☐ zusätzliche Hilfe war notwendig

Ende: Uhr

Wählen Sie als *Startscene* der *Exhibition* die als erstes angelegten *Scene*.

Aufgabe 14: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar

Beginn: Uhr ☐ zusätzliche Hilfe war notwendig

Ende: Uhr

Erstellen Sie für die zweite *Scene* einen *Exhibition Module Link*. Geben Sie ihm einen Titel und verbinden Sie den *Exhibition Module Link* mit dem erstellten *Exhibition Module*.

Aufgabe 15: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar

Beginn: Uhr ☐ zusätzliche Hilfe war notwendig

Ende: Uhr

Löschen Sie den *Scene Link* in der ersten *Scene*, der keine *Scene* verlinkt.

Aufgabe 16:

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ nicht lösbar

☐ zusätzliche Hilfe war notwendig

Beginn: Uhr

Ende: Uhr

Erstellen Sie die HTML-Seiten für die von Ihnen erstellte Virtuelle Ausstellung.

Abschließend noch einige Fragen zur Anwendung allgemein.

1. Fanden Sie die Anwendung intuitiv nutzbar?

☐ ja, sehr intuitiv ☐ ja, teilweise ☐ nein, größtenteils nicht ☐ nein, überhaupt nicht intuitiv

2. Fanden Sie die Anwendung schnell erlernbar?

☐ ja, sehr schnell erlernbar ☐ ja, teilweise ☐ nein, größtenteils nicht ☐ nein, nicht erlernbar ohne intensive Einarbeitung

3. Fanden Sie die Symbole und Beschriftung der Bedienungselemente aussagekräftig?

☐ ja, sehr aussagekräftig ☐ ja, teilweise ☐ nein, größtenteils nicht ☐ nein, überhaupt nicht aussagekräftig

4. Haben Sie die benötigten Menueinträge in dem Menu gefunden wo Sie sie erwartet haben?

☐ ja ☐ ja, teilweise ☐ nein, größtenteils nicht ☐ nein, überhaupt nicht

Verbesserungsvorschläge und –ideen:

Vielen Dank für Ihre Teilnahme!

E.2. Auswertung der Anwendertests

Im diesem Abschnitt befinden sich einige Diagramme zur Auswertung der Anwendertests.

E.2.1. Bearbeitungszeit der Aufgaben

Abbildung E-1 zeigt die Bearbeitungszeiten der Testpersonen für die Aufgaben des Anwendertests. Im Durchschnitt betrug die Länge der Bearbeitungszeit etwa 28 Minuten.

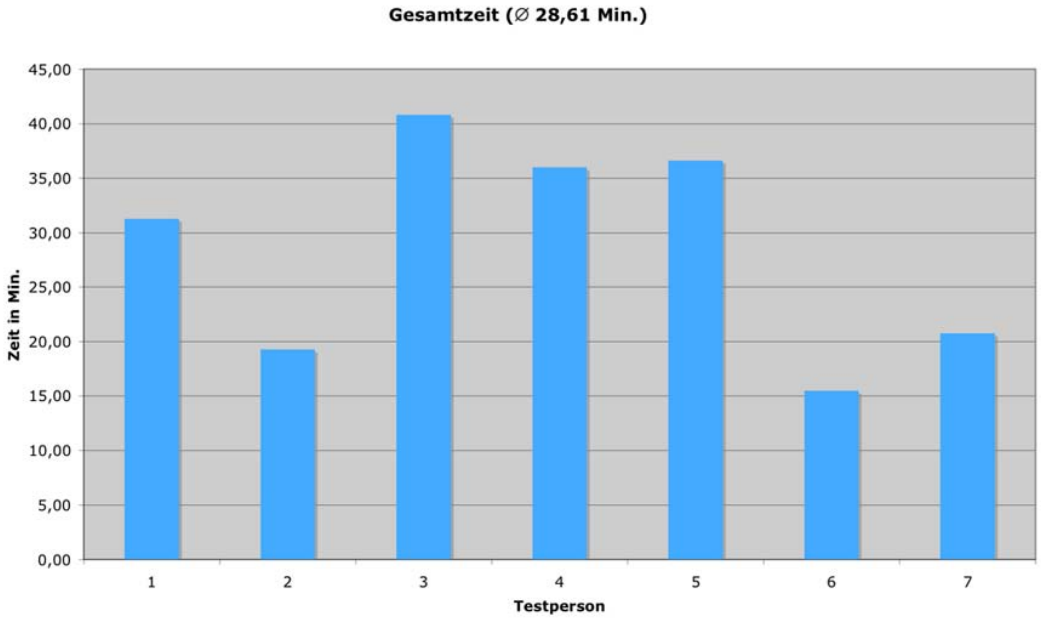


Abb. E-1. Bearbeitungszeit der Aufgaben

E.2.2. Schwierigkeit der Aufgabe

Abbildung E-2 zeigt die durchschnittlichen Einschätzungen der Schwierigkeit der Aufgaben durch die Testpersonen.

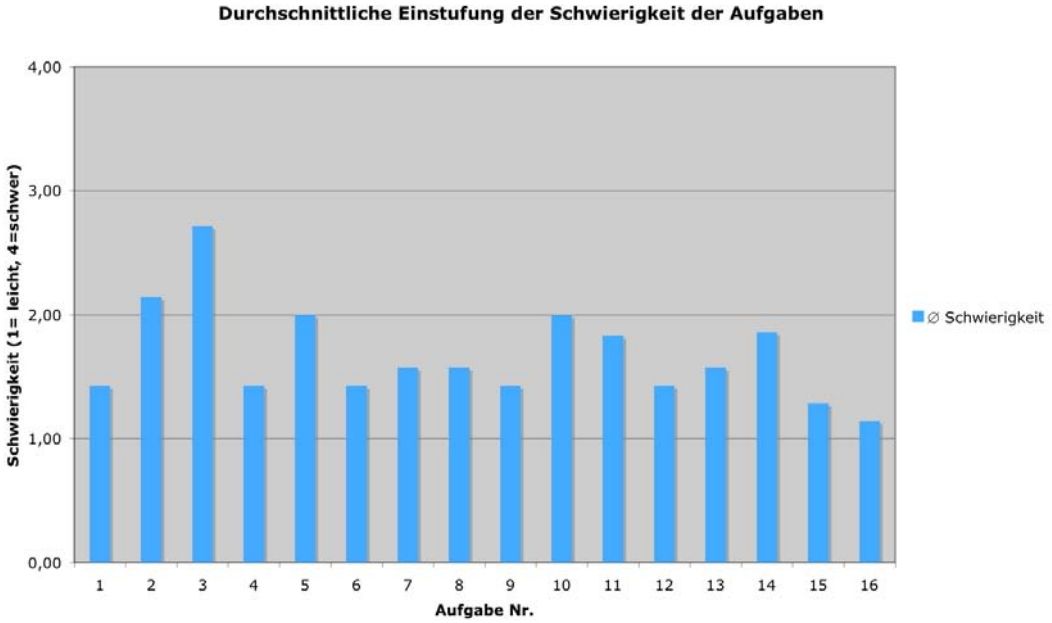


Abb. E-2. Schwierigkeit der Aufgaben

E.2.3. Benötigte Hilfe zur Bearbeitung der Aufgaben

Abbildung E-3 zeigt die von den Testpersonen benötigte Hilfe zur Bearbeitung der Aufgaben des Anwendertests in Prozent.

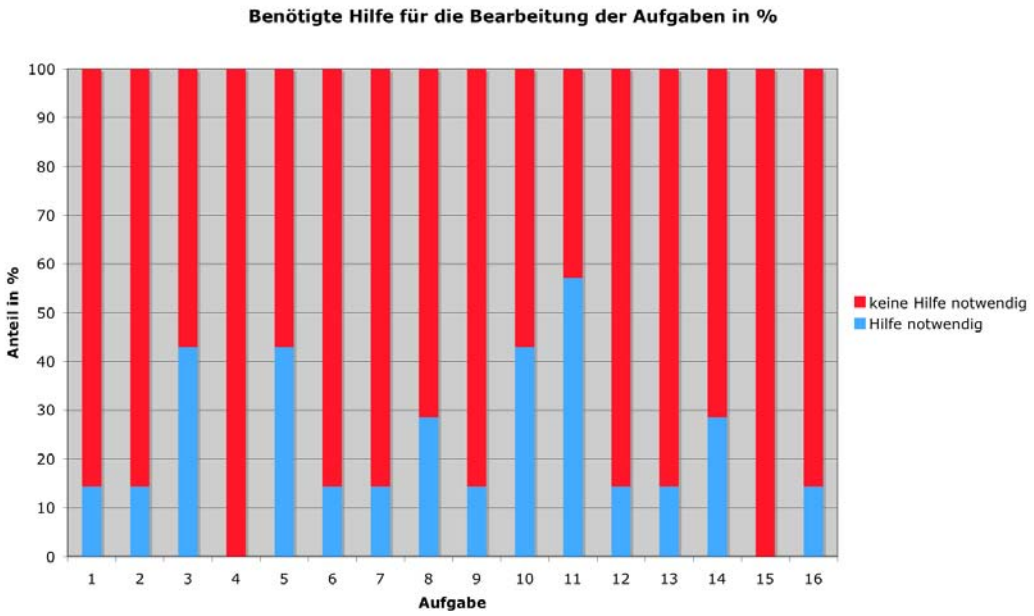


Abb. E-3. Benötigte Hilfe zur Bearbeitung der Aufgaben

E.2.4. Fragen zur Software-Ergonomie

Abbildung E-4 zeigt als wie intuitiv die Testpersonen den Prototyp eingeschätzt haben. Die prozentualen Angaben beziehen sich auf die Gesamtheit aller gegebenen Antworten.

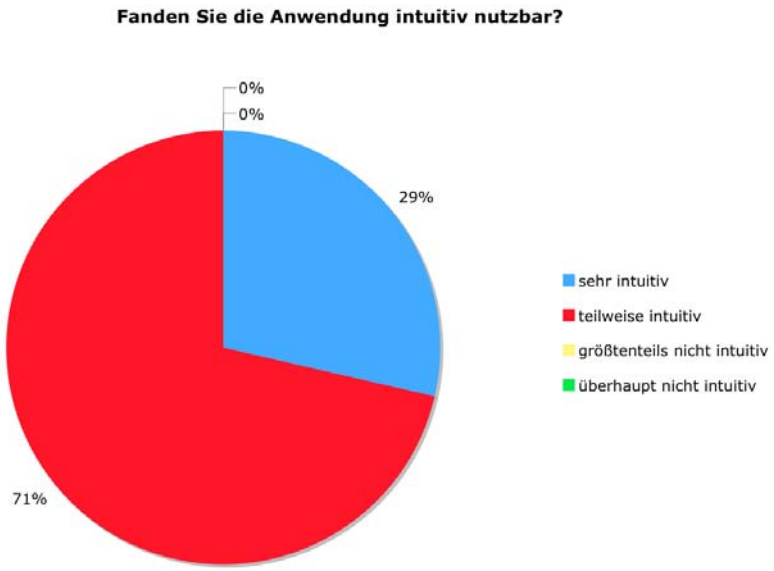


Abb. E-4. Intuitive Bedienung des Prototyps

Abbildung E-5 zeigt in wie weit die Testpersonen die Menüeinträge dort vorfanden, wo sie sie erwartet haben. Die prozentualen Angaben beziehen sich ebenfalls auf die Gesamtheit aller gegebenen Antworten.

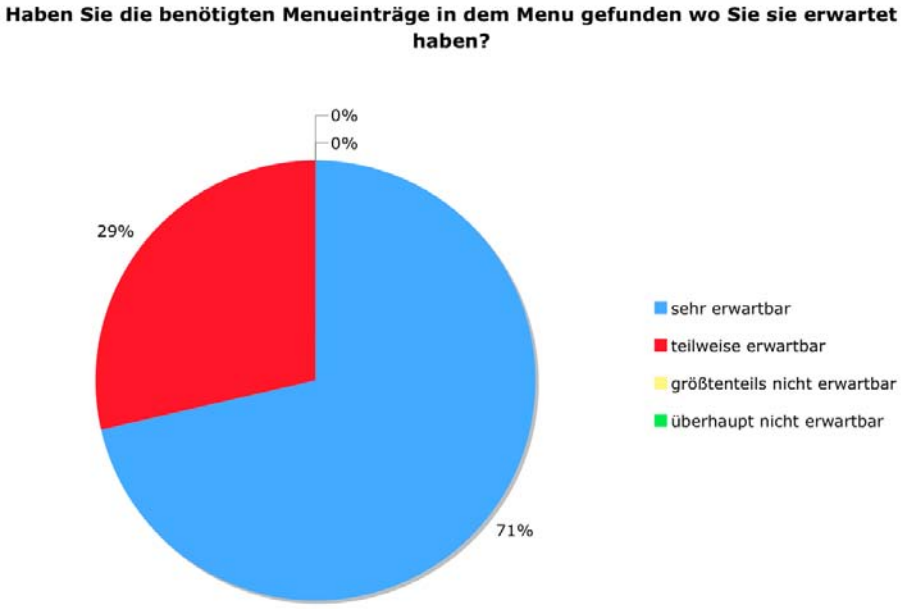


Abb. E-5. Erwartetes Vorfinden der Menüeinträge

Abbildung E-6 zeigt wie die Testpersonen die Aussagekraft der Symbole und Beschriftungen der Bedienungselemente empfanden. Die prozentualen Angaben beziehen sich ebenfalls auf die Gesamtheit aller gegebenen Antworten.

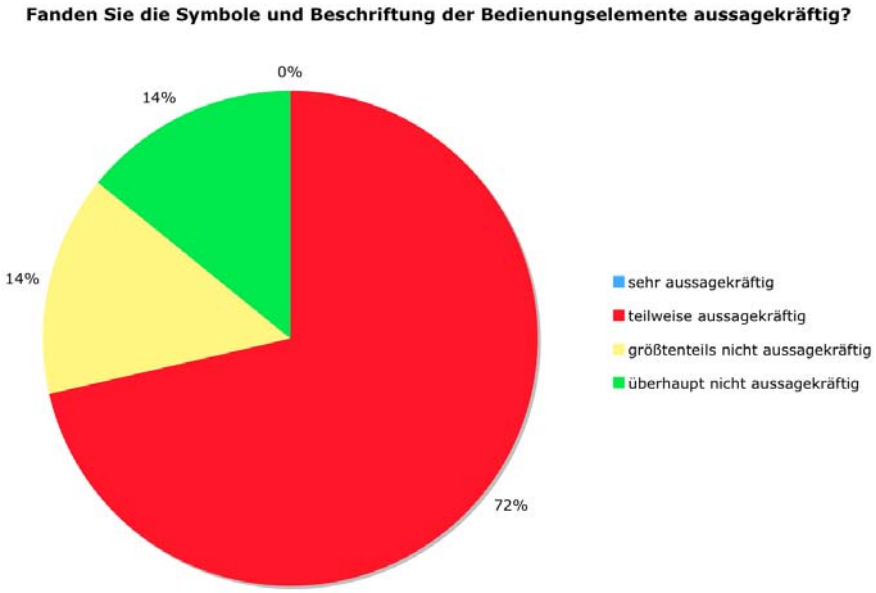


Abb. E-6. Aussagekraft von Symbolen und Beschriftungen

Anhang F.

Inhalt der CD-ROM

Nachstehend ist der Inhalt der der Diplomarbeit beiliegenden CD-ROM aufgeführt. Auf der obersten Ebene der CD-ROM befinden sich dabei drei Verzeichnisse:

- `Diplomarbeit`
- `VA-Beispiel`
- `MDD-Tool`

F.1. Inhalt des Verzeichnisses `Diplomarbeit`

- Schriftliche Ausarbeitung der Diplomarbeit
  `Diplomarbeit_Damerow.pdf`
- Für die Ausarbeitung verwendete Web-Quellen
  `Web-Quellen/*`

F.2. Inhalt des Verzeichnisses `VA-Beispiel`

Dieses Verzeichnis enthält ein Beispiel für die Website einer virtuellen Ausstellung, die mittels des Prototyps erstellt wurde.

F.3. Inhalt des Verzeichnisses `MDD-Tool`

Dieses Verzeichnis enthält zum einen die Quelltexte des Prototyps bestehend aus mehreren Eclipse-Projekten.

- Projekt, welches das Metamodell und die GMF-Modelle enthält
  `src/JAT`

- EMF.Edit-Projekt, das unter anderem die Adapter für die Metamodellelemente enthält
  `src/JAT.edit`
- Hauptprojekt des Prototyps
  `src/org.jat.exhibition.diagram`
- Plugin zur Unterstützung des projektübergreifenden Umgangs mit Dateien
  `src/org.jat.filehandler`
- Plugin zur Codegenerierung
  `src/org.jat.generation`
- Plugin zur Codegenerierung: Template-Engine
  `src/org.jat.generation.engine`
- Plugin für die Unterdiagramme des Prototyps
  `src/org.jat.mediastation.diagram`
- Plugin für die Template-Verwaltung
  `src/org.jat.templates`
- Plugin für die Workspace-Verwaltung
  `src/org.jat.workspace`

Des Weiteren enthält dieses Verzeichnis eine unter OS X ausführbare Eclipse RCP-Anwendung im Verzeichnis `Product/JAT`, eine Installationsanleitung für den Prototyp im Verzeichnis `Product/install.txt` und die JavaDoc des Prototyps im Verzeichnis `javadoc`.

Literaturverzeichnis

[And08] AndroMDA.org (2008): *What is andromda?* Url: `http://galaxy.andromda.org/index.php?option=com_content&task=blogcategory&id=0&Itemid=42`. Besucht am 9. Juni 2008.

[Atk03] Atkinson, C. und Kühne, T. (2003): *Model-driven development: a metamodeling foundation.* In: *IEEE Software*, 20(5), 36 – 41.

[Ave06] Avery, J. und Holmes, J. (2006): *Windows Developer Power Tools.* O'Reilly. ISBN 978-0-59-652754-9.

[Bal05] Balzert, H. (2005): *UML 2 kompakt - mit Checklisten.* 2. Auflage, Elsevier, Spektrum Akademischer Verlag.

[Bar03] Barnert, S., Boeckh, M., Delbrück, D. M., Greulich, W., Heinisch, C., Karcher, R., Lienhart, K., Radons, D. G., Voets, S. und Wallenwein, K. (2003): *Der Brockhaus - Computer und Informationstechnologie.* F. A. Brockhaus. ISBN 3-7653-0251-1.

[Bau06] Bauer, C. und King, G. (2006): *Java Persistence with Hibernate.* Manning. ISBN 978-1-932-39488-7.

[Bha06] Bhatia, N. (2006): *Introduction.* Url: `http://galaxy.andromda.org/index.php?option=com_content&task=view&id=104&Itemid=89`. Letzte Änderung am 25. November 2006, besucht am 9. Juni 2008.

[Car05] Carlson, D. (2005): *Eclipse Distilled.* Addison Wesley Professional. ISBN 978-0-321-28815-8.

[Cla99] Clark, James (W3C) (1999): *Xsl transformations (xslt).* Url: `http://www.w3.org/TR/xslt`. Letzte Änderung am 16. November 1999, besucht am 22. Mai 2008.

[Cli98] Cline, M., Lomow, G. und Girou, M. (1998): *C++ FAQs.* 2. Auflage, Addison Wesley Professional. ISBN 978-0-201-30983-6.

[Coc06] Cockburn, A. (2006): *Agile Software Development: The Cooperative Game, Second Edition.* 2. Auflage, Addison Wesley Professional. ISBN 978-0-321-48275-4.

[Cua07] Cuadrado, J. S. und Molina, J. G. (2007): *Building domain-specific languages for model-driven development.* In: *IEEE Software*, 24(5), 48 – 55.

[Ecl05a] Eclipse Foundation (2005): *Developer guide to diagram runtime framework.* Url: `http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.gmf.doc/prog-guide/index.html`. Besucht am 28. Mai 2008.

[Ecl05b] Eclipse Foundation (2005): *The eclipse modeling framework (emf) overview.* Url: `http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.emf.doc//references/overview/EMF.html`. Letzte Änderung am 16. Juni 2005, besucht am 27. Mai 2008.

[Ecl06] Eclipse Foundation (2006): *Common jet tags.* Url: `http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.jet.doc/gettingStarted/commonTags.xhtml`. Besucht am 4. Juni 2008.

[Ecl07a] Eclipse Foundation (2007): *Draw2d overview.* Url: `http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.draw2d.doc.isv/guide/overview.html`. Besucht am 26. Juni 2008.

[Ecl07b] Eclipse Foundation (2007): *Gef programmer's guide.* Url: `http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.gef.doc.isv/guide/guide.html`. Besucht am 28. Mai 2008.

[Ecl08a] Eclipse Foundation (2008): *Eclipse newcomers faq.* Url: `http://www.eclipse.org/home/newcomers.php`. Besucht am 24. Juni 2008.

[Ecl08b] Eclipse Foundation (2008): *Jet faq how do i create user specific code regions?* Url: `http://wiki.eclipse.org/JET_FAQ_How_do_I_create_user_specific_code_regions%3F`. Besucht am 5. Juni 2008.

[Ecl08c] Eclipse Foundation (2008): *Jet faq how do i use jmerge to manage user specific code?* Url: `http://wiki.eclipse.org/JET_FAQ_How_do_I_use_JMerge_to_manage_user_specific_code%3F`. Besucht am 5. Juni 2008.

[Ecl08d] Eclipse Foundation (2008): *Jet faq what is jmerge?* Url: `http://wiki.eclipse.org/JET_FAQ_What_is_JMerge%3F`. Besucht am 5. Juni 2008.

[Ecl08e] Eclipse Foundation (2008): *Model to text (m2t).* Url: `http://www.eclipse.org/modeling/m2t/`. Besucht am 2. Juni 2008.

[Eff08] Efftinge, S., Friese, P., Haase, A., Hübner, D., Kadura, C., Kolb, B., Köhnlein, J., Moroff, D., Thoms, K., Völter, M., Schönbach, P. und Eysholdt, M. (2008): *oaw tutorial.* Url: `http://www.eclipse.org/gmt/oaw/doc/4.3/html/contents/emf_tutorial.html`. Letzte Änderung am 11. April 2008, besucht am 6. Juni 2008.

[Fow02] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R. und Stafford, R. (2002): *Patterns of Enterprise Application Architecture.* Addison Wesley Professional. ISBN 978-0-321-12742-6.

[Fow05] Fowler, M. (2005): *Language workbenches: The killer-app for domain specific languages?* Url: `http://martinfowler.com/articles/languageWorkbench.html`. Letzte Änderung am 12. Juni 2005, besucht am 13. Mai 2008.

[Gam94] Gamma, E., Helm, R., Johnson, R. E. und Vlissides, J. M. (1994): *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional. ISBN 978-0201633610.

[Gam03] Gamma, E. und Beck, K. (2003): *Contributing to Eclipse: Principles, Patterns, and Plug-Ins.* Addison Wesley Professional. ISBN 978-0-321-20575-9.

[Gov07] Govoni, R. (2007): *Use eclipse jet to automate model-driven development aspects.* Url: `http://www.devx.com/opensource/Article/34929?trk=DXRSS_LATEST`. Letzte Änderung am 3. Juli 2007, besucht am 2. Juni 2008.

[Hes94] Hesse, W., Barkow, G., von Braun, H., Kittlaus, H.-B. und Scheschonk, G. (1994): *Terminologie der softwaretechnik.* In: *Informatik-Spektrum*, 17, 39–47.

[Hol04] Holzner, S. (2004): *Eclipse.* 1. Auflage, O'Reilly. ISBN 3-89721-385-0.

[Hör03] Hörtzsch, M. und Rapko, A. (2003): *Content Management und Zope.* 1. Auflage, mitp. ISBN 3-8266-1336-8.

[Ing07] Ing, D., Gronback, R., P., Y. und Reitsma, J. (2007): *Gmf tutorial.* Url: `http://wiki.eclipse.org/index.php/GMF_Tutorial`. Letzte Änderung am 20. November 2007, besucht am 29. Mai 2008.

[Int08] Interactive Objects (2008): *Arcstyler – the leading development platform for mda.* Url: `http://www.interactive-objects.com/fileadmin/pdf/products/as_5_5_factsheet.pdf`. Besucht am 30. Mai 2008.

[Knu05] Knudsen, J. und Niemeyer, P. (2005): *Learning Java.* 3. Auflage, O'Reilly. ISBN 0-596-00873-2.

[Lar01] Larman, C. (2001): *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.* 2. Auflage, Prentice Hall. ISBN 978-0-13-092569-5.

[Max07] Max-Planck-Institut für Wissenschaftsgeschicht (2007): *Max-planck-institut für wissenschaftsgeschichte - profil.* Url: `http://www.mpiwg-berlin.mpg.de/de/institut/index.html`. Besucht am 17. Juni 2008.

[Mel04] Mellor, S. J., Scott, K., Uhl, A. und Weise, D. (2004): *MDA Distilled: Principles of Model-Driven Architecture.* Addison Wesley Professional. ISBN 978-0-201-78891-4.

[Mil06] Miles, R. und Hamilton, K. (2006): *Learning UML 2.0.* O'Reilly. ISBN 978-0-59-600982-3.

[Nie02] Niederst, J. (2002): *Webdesign in a Nutshell.* 1. Auflage, O'Reilly. ISBN 3-89721-294-3.

[Nix05] Nix, M. (2005): *Web Content Management. CMS verstehen und auswählen.* 1. Auflage, Entwickler.Press. ISBN 978-3935042642.

[OMG05] OMG (Object Management Group) (2005): *Unified modeling language specification.* Url: `http://www.omg.org/cgi-bin/doc?formal/05-04-01`. Letzte Änderung am 4. Juni 2005, besucht am 26. Mai 2008.

[OMG06] OMG (Object Management Group) (2006): *Meta object facility (mof) 2.0 core specification - chapter 12 the essential mof (emof) model.* Url: `http://www.omg.org/docs/html/06-01-01/Output/12_EMOF.html#34605`. Letzte Änderung am 1. Januar 2006; Besucht am 3. August 2008.

[OMG07] OMG (Object Management Group) (2007): *About the object management group.* Url: `http://www.omg.org/gettingstarted/gettingstartedindex.htm`. Letzte Änderung am 9. November 2007, besucht am 9. Mai 2008.

[ope08] openArchitectureWare.org (2008): *openarchitectureware - official openarchitectureware homepage.* Url: `http://www.openarchitectureware.org/`. Besucht am 6. Juni 2008.

[Pea04] Pearson Education, Inc. (2004): *Faq pages, parts, sites, windows: What is all this stuff?* Url: `http://wiki.eclipse.org/FAQ_Pages%2C_parts%2C_sites%2C_windows:_What_is_all_this_stuff%3F`. Besucht am 4. Juli 2008.

[Pem00] Pemberton, Steven (W3C) et al. (2000): *Xhtml^{TM} 1.0 the extensible hypertext markup language (second edition).* Url: `http://www.w3.org/TR/xhtml1/`. Letzte Änderung am 26. Januar 2000; Besucht am 4. August 2008.

[Pet06] Petrasch, R. und Meimberg, O. (2006): *Model Driven Architecture - Eine praxisorientierte Einführung in die MDA.* 1. Auflage, dpunkt.verlag. ISBN 3-89864-343-3.

[Pop04] Popma, R. (2004): *Jet tutorial part 1 (introduction to jet).* Url: `http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html`. Letzte Änderung am 31. Mai 2004; Besucht am 29. Juli 2008.

[Ray01] Ray, E. T. (2001): *Einführung in XML.* 1. Auflage, O'Reilly.

[Ric05] Richter, S. (2005): *Zope 3 Developer's Handbook.* Sams. ISBN 978-0-672-32617-2.

[Sch96] Schiedermeier, P. D. R. (1996): *Klassifikation von fehlern.* Url: `http://www.mi.uni-koeln.de/c/mirror/f7alpha1.informatik.fh-muenchen.de/~schieder/programmieren-1-ws96-97/fehlerarten.html#section_10_1_3`. Letzte Änderung am 10. Oktober 1996, besucht am 9. Mai 2008.

[Sch99] Schefe, P. (1999): *Softwaretechnik und erkenntnistheorie.* In: *Informatik-Spektrum*, 22(2), 122–135.

[Spa07] Sparx Systems (2007): *Enterprise architect with model driven architecture.* Url: `http://www.sparxsystems.com/bin/MDA%20Tool.pdf`. Besucht am 31. Mai 2008.

[Sta07] Stahl, T., Völter, M., Efftinge, S. und Haase, A. (2007): *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management.* 2. Auflage, dpunkt.verlag. ISBN 978-3-89864-448-8.

[Ste03] Steinberg, F. B. D., Merks, E., Ellersick, R. und Grose, T. J. (2003): *Eclipse Modeling Framework: A Developer's Guide.* Addison Wesley Professional. ISBN 978-0-13-142542-2.

[Sun06] Sun Microsystems, Inc. (2006): *Frequently asked questions about java-server faces technology.* Url: `http://wiki.java.net/bin/view/Projects/JavaServerFacesSpecFaq`. Letzte Änderung am 11. Juli 2006; Besucht am 3. August 2008.

[Sun08] Sun Microsystems, Inc. (2008): *Java ee faq.* Url: `http://java.sun.com/javaee/overview/faq/javaee_faq.jsp`. Besucht am 23. Mai 2008.

[Tro07] Trompeter, J., Pietrek, G., Beltran, J. C. F., Holzer, B., Kamann, T., Kloss, M., Mork, S. A., Niehues, B. und Thoms, K. (2007): *Modellgetriebene Softwareentwicklung - MDA und MDSD in der Praxis.* entwickler.press.

Glossar

Abstrakte Syntax

Hier: beschreibt die Metamodellelemente und ihre Beziehungen untereinander.

AndroMDA

Open-Source-Generator-Framework

Applikationsserver

Auch *Application Server*; ein Server, der über ein Netzwerk Anwendungsprogramme zur Verfügung stellt. ([Bar03], S. 63)

Betatest

Letzte Testphase eines Produktes bevor es ausgeliefert wird.

Branching Point

Spezielle Ausprägung eines Slides.

Bug

Fehler in der Programmierung einer Software

Bytecode

Zwischencode, in den ein Quelltext übersetzt wird, der in der Regel prozessorunabhängig ist; wird von einer virtuellen Maschine ausgeführt.

Cartridge

Modul eines Generators zur Kapselung von speziellen Teilen des Zielsystems.

Compiler

Programm, das Quelltext in Maschinensprache übersetzt. ([Bar03], S. 182)

Constraints

Bedingungen, die erfüllt sein müssen, damit ein Modell wohlgeformt ist.

Coolbar

Hier: zweite Menüleiste einer Eclipse RCP; enthält Buttons, die häufig gebrauchte Funktionen breitstellen.

Diagramm

Hier: Darstellung eines Modells oder eines Teils eines Modells im Editor eines GMF-Modellierungstools.

Domäne

Begrenztes Wissens- oder Interessengebiet.

Domänenmodell

Visuelle Repräsentation der Elemente einer Domäne und deren Beziehungen untereinander.

Dynamische Semantik

Legt die erlaubten Ausführungen eines Ausdrucks fest.

Eclipse

In Java implementierte Entwicklungsumgebung für jegliche Art von Software, die durch Plugins erweiterbar ist.

Editor

Hier: der Teil eines GMF-Modellierungstools, der der Modellierung dient; Allgemein: Anwendung zur Erstellung und Bearbeitung von Daten.

Entwurfsmuster

Auch: Pattern; Eine *«bewährte, generische Lösung für ein immer wiederkehrendes Entwurfsproblem* [...]*, das in bestimmten Situationen auftritt»* ([Bal05], S. 74).

Exhibition Module

Stellt dem Besucher einer virtuellen Ausstellung Informationen zum Thema einer Scene zur Verfügung.

Exhibition Module Link

Verweis der eine Scene mit einem Exhibition Module verbindet.

formal

Gemäß einem mathematisch exaktem, widerspruchsfreiem und vollständig definiertem Regelwerk.

Framework

Ein Framework ist ein Programmgerüst, bestehend aus unterschiedlichen Klassen, die miteinander interagieren. *«The framework defines a generic program structure that is suitable for building a group of related applications or systems. Usually the framework provides default behavior for common situations while allowing the default behavior to be easily overridden with application-specific behavior when necessary. That is, the framework is designed to be customized for solving a specific problem or building a specific business application.»* ([Cli98], FAQ 4.08)

Generator

Auch: Codegenerator; Software, die den Generierungsprozess durchführt.

Generierungsprozess

Auch: Transformationsprozess; beinhaltet das Einlesen, Validieren und Transformieren eines Modells; wird durch einen Generator durchgeführt.

GNU Lesser General Public License

Eine von der Free Software Foundation entwickelte Lizenz für die Lizensierung freier Software.

Google

Internet-Suchmaschine

Interpreter

Software, die Modelle zur Laufzeit auswertet und darauf basierend Aktionen ausführt.

J2EE

Java 2 Enterprise Edition; ein Industriestandard zur Entwicklung von server-seitigen Java Applikationen. ([Sun08])

Java

Objektorientierte höhere Programmiersprache von der Firma Sun Microsystems.

Java Virtual Machine

abgekürzt JVM; Teil der Java-Laufzeitumgebung; führt den Java Bytecode aus.

JavaScript

Hauptsächlich in Webbrowsern eingesetzte Skriptsprache.

JBoss

Open-Source-Application Server von RedHat

JUnit

Von Erich Gamma und Kent Beck entwickeltes Framework zum Testen von Java Programmen. Mehr Information sind unter http://www.junit.org/ zu finden.

Konkatenation

In der Informatik wird hierunter die Verkettung von einzelnen Zeichen oder Zeichenketten verstanden.

Konkrete Syntax

Hier: legt die Ausdrucksmittel zur Beschreibung eines Modells fest.

Look and Feel

Aussehen und Handhabung einer Software; bestimmt durch beispielsweise die verwendete Schriftart, die verwendeten Farben etc.

Meta-Sequence

Legt die Reihenfolge fest, in der Sequences angezeigt werden.

Metametamodell

Metamodell eines Metamodells.

Metamodell

Formale Beschreibung einer Domäne, die Aussagen über deren Struktur macht.

Middleware

«Software für den objektorientierten, direkten Datenaustausch zwischen Anwendungsprogrammen.» ([Bar03], S. 580)

Modell

Hier: Abbild eines Softwaresystems.

Modellierungsfehler

Verstöße gegen die abstrakte Syntax oder die Constraints.

Modellvalidierung

Auch: Validierung; Überprüfung, ob die abstakte Syntax und die Constraints vom Modell eingehalten werden.

Modulares Metamodell

Metamodell, das aus mehreren Metamodellen besteht.

OS X

Betriebssystem der Firma Apple

parsen

Die Analyse einer beliebigen Eingabe und deren Umwandlung in eines anderes Format.

Plattform

Hier: Hard- und Software-Komponenten eines Rechners, sowie Middleware und Programmbibliotheken.

plattformunabhängig

Hier: unabhängig von der Hard- und Software eines Rechners und unabhängig von einer Programmiersprache oder Middleware.

PowerPoint

Ein Programm von Microsoft zur Erstellung und Vorführung von Präsentationen. ([Bar03], S. 725)

Protected Region

Geschützter Bereich im Quelltext, der bei einer erneuten Generierung nicht überschrieben wird.

Python

Eine interpretierte, objektorientierte Programmiersprache für Unix, Windows und Mac OSX. [S. 63]Brockhaus

Reguläre Ausdrücke

auch *Regular Expressions*; ein regulärer Ausdruck ist eine Zeichenkette, die der Beschreibung von komplexen Zeichenmustern unter Verwendung von speziellen syntaktischen Regeln dient.

Rendern

Unter Rendern wird hier (X)HTML-Rendering verstanden. Beim Rendern wird ein (X)HTML-Dokument visuell dargestellt. Webbrowser beispielsweise rendern HTML- und XHTML-Dokumente, um sie anzuzeigen. Das Rendern wird durch die *Rendering-Engine* übernommen. Der Webbrowser Firefox beispielsweise nutzt die Rendering-Engine *Gecko*.

Ruby

Dynamisch getypte Programmiersprache, die ursprünglich von Yukihiro "matz" Matsumoto entwickelt wurde.

Scene

Abschnitt einer Virtual Exhibition.

Scene Link

Verweis der zwei Scenes verbindet.

Sequence

Legt die Reihenfolge fest, in der Slides und Branching Points angezeigt werden.

Slide

Kleinste Einheit eines Exhibition Module.

Statische Semantik

Legt die Bedingungen fest, die ein Ausdruck (hier: ein Modell) erfüllen muss um wohlgeformt zu sein.

Tag

Platzhalter in einem Template.

Template

Vorlage zur Erzeugung von Dateien.

Template-Engine

Software zur Erstellung von Dateien anhand von Templates.

Template-Sprache

Sprache zur Erstellung von Templates; legt beispielsweise fest, wie Tags notiert werden.

Transformationssprache

Sprache zur Beschreibung von Transformationen.

Transformationswerkzeug

Werkzeug für die Definition und Durchführung von Transformationen.

Validator

Software oder Teil einer Software, der die Modellvalidierung durchführt.

Versionskontrollsystem

System zur Unterstützung der Versionierung und des gemeinsamen Zugriffs auf in der Regel Quelltext-Dateien.

Virtual Exhibition

Auch: virtuelle Ausstellung; virtueller Rundgang, der einen realen oder abstrakten Raum abbildet.

Web-Framework

Framework, das die Entwicklung von dynamischen Websites und Web-Anwendungen unterstützt.

Webbrowser

Ein Anwendungsprogramm für das Aufrufen und Anzeigen von Dokumenten im Internet. ([Bar03], S. 966)

Windows

Betriebsystem der Firma Microsoft

wohlgeformt

Auch: valide; ein Ausdruck ist wohlgeformt, wenn alle Regeln, die die statische Semantik festlegt, erfüllt sind.

XHTML

Extensible HTML; XML-basierte Neuimplementierung von HTML.

XHTML-Renderer

Software zur Darstellung von XHTML-Dokumenten.

XPath

XML Path Language; Sprache zur Adressierung von Teilen eines XML-Dokumentes. ([Ray01], S. 339)

Xtext

Framework zur Definition von textuellen DSLs; Teil des oAW-Projekts.

Zielplattform

Plattform, für die eine Transformation spezifisch ist.

Zope

Open-Source-Applikationsserver, der in der Programmiersprache Python entwickelt wurde.

Abkürzungsverzeichnis

CMS (Content Management System)

Speziell konzipierte Software zur Erstellung und Verwaltung von Text- und Multimedia-Dokumenten.

CORBA (Common Object Request Broker Architecture)

Ein *«von der OMG entwickelter Industriestandard für die Kommunikation zwischen objektorientierten Software-Komponenten, die in einem Netzwerk verteilt sind.»* ([Bar03], S. 580)

DHTML (Dynamic HTML)

Variante der HTML, die es ermöglicht, dass ein Dokument sich nach der Anzeige im Browser noch ändert.

DSL (Domain Specific Language)

Auch *domänenspezifische Sprache*; speziell auf eine Domäne abgestimmte Sprache.

EMF (Eclipse Modeling Framework)

Plugin für Eclipse, das ein Java-Framework zur Erzeugung von Modellierungswerkzeugen bereitstellt.

EMOF (Essential MOF)

Reduzierte Version der MOF.

GEF (Graphical Editing Framework)

Plugin für Eclipse zur graphischen Darstellung und Bearbeitung von Modellen.

GME (Generic Modeling Environment)

Programm zur Erstellung von domänenspezifischen Modellierungstools.

GMF (Graphical Modeling Framework)

Plugin für Eclipse zur Entwicklung von graphischen Editoren zur Erstellung und Persistierung von Modellen.

HTML (Hypertext Markup Language)

Beschreibungssprache für Dokumente zur Veröffentlichung dieser im Internet.

IDE (Integrated Development Environment)

Auch: integrierte Entwicklungsumgebung; Anwendung zur Entwicklung von Software.

JAT (JAT Ausstellungs Tool)

Projektbezeichnung des im Rahmen dieser Diplomarbeit entwickelten Prototyps.

JDK (Java SE Development Kit)

Beinhaltet die Java-Laufzeitumgebung und unterschiedliche Java-Entwicklungswerkzeuge.

JDT (Java Development Tools)

Plugins für Eclipse zur Entwicklung von Java.

JET (Java Emitter Templates)

Codegenerator; Teil des M2T-Projekts.

JRE (Java Runtime Environment)

Laufzeitumgebung für die Programmiersprache Java

JSF (JavaServer Faces)

«JavaServer Faces technology is a framework for building user interfaces for web applications.» ([Sun06])

JSP (JavaServer Pages)

Eine *«von* [...] *Sun Microsystems entwickelte Server-seitige Programmiersprache zur Entwicklung von plattformunabhängigen Internetanwendungen»* ([Bar03], S. 491); mittels JSP können Templates erstellt werden, die der automatischen Erzeugung von HTML- und XML-Dateien dienen.

MDA (Model Driven Architecture)

Standard der OMG zur modellgetriebenen Softwareentwicklung.

MDD (Model Driven Development)

Modellgetriebene Softwareentwicklung

MDSD (Model Driven Software Development)

Modellgetriebene Softwareentwicklung

MIC (Model-Integrated Computing)

Spezielle Ausprägung der modellgetriebenen Softwareentwicklung im Umfeld von Echtzeitsystemen und eingebetteten Systemen.

MOF (Meta Object Facility)

Metametamodell, das durch die OMG standardisiert ist.

MPG (Max-Planck-Gesellschaft)

Forschungsgesellschaft

MPIWG (Max-Planck-Institut für Wissenschaftsgeschichte)

Forschungsinstitut der MAX-PLANCK-GESELLSCHAFT

M2C-Transformation (Modell-zu-Code-Transformation)

Auch: Modell-zu-Text-Transformation, Codegenerierung oder Generierung; Automatische Erzeugung von Text aus einem Modell.

M2M-Transformation (Modell-zu-Modell-Transformation)

Automatische Erzeugung eines Modells aus einem anderen Modell oder die automatische Modifizierung eines Modells.

M2T (Model To Text)

Projekt von Eclipse; befasst sich mit der Generierung von Textdokumenten aus Modellen.

oAW (openArchitectureWare)

Leistungsstarkes, in Java implementiertes Generator-Framework.

OCL (Object Constraint Language)

Durch die OMG standardisierte Sprache, die unter anderem zur Definition von Constraints verwendet werden kann.

OMG (Object Management Group)

Internationales Konsortium, das sich mit der Entwicklung von Technologie-Standards beschäftigt. Die OMG wurde 1989 gegründet. ([OMG07])

PDM (Platform Description Model)

Modell bei MDA; beschreibt die Struktur der Plattform.

PIM (Platform Independent Model)

Modell bei MDA; beschreibt Anwendungslogik ohne plattformspezifische Details.

PSM (Platform Specific Model)

Modell bei MDA; wird aus PIM und PDM erzeugt; enthält plattformspezifische Details.

QVT (Query, Views, and Transformations)

Standard der OMG für Modelltransformationen

Eclipse RCP (Eclipse Rich-Client-Platform)

Mit Eclipse entwickelbare Rich-Client-Anwendung, die dem Plugin-Konzept folgt.

SVN (Subversion)

Open-Source-Versionskontrollsystem

SWT (Standard Widget Toolkit)

Klassenbibliothek von IBM zur Entwicklung von graphischen Benutzeroberflächen, die dem Look and Feel des verwendeten Betriebssystems entsprechen.

TPTP (Test & Performance Tools Platform)

Projekt von Eclipse, das unterschiedliche Teilprojekte zum Testen von Software und deren Performance umfasst.

UML (Unified Modeling Language)

Durch die OMG standardisierte Sprache zur Modellierung von unter anderem Softwaresystemen.

WYSIWYG (What You See Is What You Get)

Prinzip zur Bearbeitung von Dokumenten; hierbei wird ein Dokument bei der Bearbeitung so am Bildschirm angezeigt, wie es bei einer späteren Ausgabe, zum Beispiel beim Druck, aussieht.

XMI (XML Metadata Interchange)

Austauschformat für Modelle, die auf der MOF basieren.

XML (Extensible Markup Language)

Eine *«Sprache, mit der die Struktur von Dokumenten beschrieben wird»* ([Bar03], S. 992).

XSD (XML Schema Definition)

XML-Dokument, zur Definition der Struktur von XML-Dokumenten.

XSLT (Extensible Stylesheet Language Transformation)

Eine Transformationssprache für die Umwandlung von XML-Dokumenten in andere XML-Dokumente ([Cla99]).

ZMI (Zope Management Interface)

Web-Interface einer Zope-Instanz

Abbildungsverzeichnis