universität wien

# MASTERARBEIT

Titel der Masterarbeit

„Visualisierung von Metamodellen im Rahmen der Open-Model-Initiative

-

Konzeption, Design und prototypische Implementierung eines Metamodell-Browsers“


Verfasser

Peter Hirsch, Bakk.


angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Prottes, am 15. April 2011

| | |
|---|---|
| Studienkennzahl lt. Studienblatt: | A 066 926 |
| Studienrichtung lt. Studienblatt: | Masterstudium Wirtschaftsinformatik |
| Betreuer: | o. Univ.-Prof. Dr. Dimitris Karagiannis |

# Schriftliche Versicherung

Hiermit versichere ich, dass die vorliegende Diplomarbeit von mir selbständig und ausschließlich mit den angegebenen Quellen und Hilfsmitteln verfasst wurde.

Diese Arbeit wurde weder in gleicher noch in ähnlicher Form bis dato einer Prüfungsbehörde vorgelegt.

Prottes, am 15. April 2011

# Danksagung

Mein besonderer Dank gilt meiner Lebensgefährtin Ingeborg Hauser, die mich sowohl bei der Erstellung dieser Diplomarbeit als auch während des gesamten Studiums moralisch und mit guten Ratschlägen unterstützt hat. Darüber hinaus war sie es, die meine cholerischen Ausbrüche ertragen musste, wenn es wieder einmal nicht so lief, wie ich mir das vorgestellt hatte. Ebenso muss ich ihr dafür danken, dass sie sich dazu bereit erklärt hat, diese finanziell schwierige Zeit mit mir gemeinsam zu bewältigen (und glauben sie mir, es war eine schwere Zeit).

Ich möchte mich auch ganz herzlich bei den Kollegen Hanno Fallmann, Niksa Visic und Dominik Hofbauer aus der Forschungsgruppe Knowledge Engineering bedanken, die mir bei der Ausarbeitung der Diplomarbeit mit Rat und Tat zur Seite standen und mir in einer Zeit der Stagnation wichtige Impulse lieferten.

Meinem betreuenden Professor o. Univ.-Prof. Dr. Dimitris Karagiannis danke ich für das interessante Thema meiner Diplomarbeit und dafür, dass er mich auch in Motorrad-Montur in seine Sprechstunde vorgelassen hat. Darüber hinaus möchte ich mich bei ihm für die außerordentlich interessanten Lehrveranstaltungen im Zuge der letzten fünf Jahre bedanken, die mein Leben nachhaltig beeinflussen werden.

Nicht unerwähnt möchte ich meine lieben Studienkollegen- und Innen lassen, mit denen ich eine wunderschöne Zeit verleben durfte und mit denen ich es gemeinsam geschafft habe dieses Studium in so kurzer Zeit zu absolvieren.

Schlussendlich möchte ich meinen Eltern danken, die mir die intellektuellen Voraussetzungen mit auf den Weg gegeben haben ein solches Studium zu absolvieren und mich auch in finanziell schwierigen Zeiten nicht im Regen stehen ließen.

# Kurzfassung

Die Open-Model-Initiative hat sich zum Ziel gesetzt, eine Plattform zu etablieren, auf der (Referenz-) Modelle jedermann zur freien Verfügung angeboten werden. In Anlehnung an den Open-Source-Gedanken sollen dadurch redundante Entwicklungsprozesse verhindert und in weiterer Folge Kosten für die Entwicklung von Modellen verringert werden. Um dieses ehrgeizige Ziel umzusetzen, versucht die Open-Model-Initiative eine weitreichende Community zu etablieren, die auf einer Community-Plattform Problemstellungen der Modellierung in gemeinschaftlicher und verteilter Arbeit löst.

Jedes Modell basiert auf einer Modellierungssprache, welche wiederum durch ein Metamodell beschrieben wird. Somit steht zum Beginn eines jeden Modellentwicklungszyklus die Wahl einer adäquaten Modellierungssprache. Um einen Entwickler bei der Wahl der am besten passenden Modellierungssprache für eine Problemstellung zu unterstützen, behandelt diese Arbeit als Thema die Browser-gestützte Visualisierung von Metamodellen. Zu diesem Zweck wird ein Konzept und ein Prototyp erstellt, wie Metamodelle unter Verwendung eines herkömmlichen Browsers auf visuell ansprechende Weise dargestellt werden können, um Entwicklern eine Möglichkeit zu bieten, sich auf unkomplizierte Weise einen ersten Überblick über existierende Metamodelle zu verschaffen.

Diese Arbeit ist aus zwei großen Teilen aufgebaut; einer tiefergehenden theoretischen Betrachtung des wissenschaftlichen Themengebiets und einer konzeptionellen und prototypischen Entwicklung eines "Metamodell-Browsers".

Zu Beginn des theoretischen Teils wird der Modellbegriff an sich erörtert und ein erkenntnistheoretischer Bezugsrahmen erstellt. Dieser Rahmen soll garantieren, dass jeder Leser dieser Arbeit die formulierten Aussagen in nahezu gleicher Weise interpretieren kann. Danach werden die wichtigsten Begriffe und Definitionen des wissenschaftlichen Themenbereichs beschrieben und diskutiert. Anschließend wird auf die Open-Model-Initiative und deren Community eingegangen, welche die wichtigsten Adressaten dieser Arbeit darstellen. Am Ende des theoretischen Teils werden aktuelle Rich-Internet-Technologien analysiert und jene Technologie für den Prototyp ausgewählt, die am passendsten erscheint (Adobe Flex).

Der Entwicklungsteil beschreibt die konzeptionelle und prototypische Erstellung des Metamodell-Browsers. Zu Beginn wird das Oberflächendesign in einer graphischen Entwicklungsumgebung kreiert. In diesem Zusammenhang wird auch darauf eingegangen,

wie Adobe die Entwicklung einer Rich-Internet-Applikation mit Flex empfiehlt und die Vor- und Nachteile erörtert. Speziell ein Exkurs über Flash-Catalyst soll zeigen, wie diese Entwicklungs-Kette funktionieren soll. Danach wird die serverseitige Java-Logik implementiert und das Flex-Frontend unter Verwendung von Blaze-DS an den Server gekoppelt. Eine Beschreibung der Funktionalität des Metamodell-Browsers beendet den zweiten Teil.

Am Ende dieser Diplomarbeit folgen eine Zusammenfassung und ein Ausblick auf weiterführende Entwicklungsmöglichkeiten in Bezug auf den Metamodell-Browser.

# Abstract

The Open Model Initiative has the agenda to establish a platform, where (reference-)models are publicly available to everyone. Inspired by the ambition of Open-Source, the aim is to avoid redundant model development processes and thus minimizing development cost. An additional goal is to establish a versatile community which solves modelling-problems in a collaborative and distributed way.

Each model is based on a modeling language, which is described by a metamodel. In the beginning of each model development cycle an adequate modelling-language must be selected. In order to support a developer in selecting the best fitting modeling language for his problem, this master-thesis covers the subject of browser-based visualization for metamodels. To this end, a concept is created, how a metamodel can attractively be visualized on a browser and therefore provide an uncomplicated overview about the existing metamodels to developers.

This master-thesis is composed of two major parts, the closer theoretical examination of the science topic and the conceptual and prototypical development of the "Metamodel-Browser".

At the beginning of the theoretical part, an epistemological reference frame is established. This frame should guarantee that all readers of this master-thesis nearly interpret the statements in the same way. Next, the most important terms and definitions of the science topic are described and discussed, followed by a description of the Open Model Initiative and its Community, which is the most important addressee of this master-thesis. At the end of the theoretical part actual rich internet technologies are analyzed and the best fitting technology will be selected for the implementation of the prototype (Adobe Flex).

The development part describes the conceptual and prototypical construction of the Metamodel-Browser. First the application design will be modeled in a graphical tool. Next, the Adobe development chain for Rich Internet Applications is presented with a focus on the Flash-Catalyst-Tool to demonstrate how the chain works. After this, the implementation of the server side Java-logic and the client side Flex-frontend will be specified. A description of the functionality concludes the second part.

At the end of this master thesis an outlook for possible further development related to the Metamodel-Browser is provided.

# Inhaltsverzeichnis

# Tabellenverzeichnis

# Abbildungsverzeichnis

# Listings

# 1 Einleitung

## 1.1 Motivation

Informationssysteme in Wirtschaft und Verwaltung bilden einen der Schwerpunkte in den Forschungsbereichen der Wirtschaftsinformatik. Durch die zunehmende Automatisierung innerbetrieblicher Abläufe und die Verwendung neu entwickelter Technologien steigen die Anforderungen an Informationssysteme stetig. Aber nicht nur innerbetriebliche Abläufe sondern auch die Interaktionsfähigkeit zwischen zwei oder mehreren eigenständigen Betrieben führen zu immer neuen Herausforderungen. Diese können durch strategische Allianzen, durch die Fusion zweier Betriebe oder durch die Bildung eines virtuellen Unternehmens entstehen. Für die Wirtschaftsinformatik ergeben sich dadurch Aufgabenbereiche, die im speziellen die Integration verschiedener Informationssysteme und in weiterer Folge die mögliche Wiederverwendung der dafür entwickelten Artefakte betreffen. Eine mögliche Herangehensweise bildet die Referenzmodellierung, der seit geraumer Zeit großes Potential für die Bewältigung dieser Anforderungen eingeräumt wird. Durch die Referenzmodellierung ist es möglich, Artefakte und Mechanismen, die zur Anpassung von Modellen benötigt werden, anwendungsübergreifend zu beschreiben. Referenzmodelle sind sozusagen Modell-Templates, die in weiterer Folge an spezifische Sachverhalte angepasst werden können. Dadurch erfüllen sie den Anspruch der Wiederverwendbarkeit und können zu einer erheblichen Kosteneinsparung bei der Erstellung von Modellen führen. Bedient man sich zusätzlich einer einheitlichen Modellierungssprache für die Referenzmodelle, kann die Integration eines angepassten Modells in eine Informationssystemlandschaft erleichtert werden.

Seit 2006 verfolgt die Open-Model-Initiative das Ziel, angelehnt an den Open-Source-Gedanken, Referenzmodelle für jedermann zugänglich zu machen. Grundvoraussetzung für die Bereitstellung von Referenzmodellen ist eine geeignete, offene Modellierungsplattform. Auf dieser können bereits bestehende Referenzmodelle von den Mitgliedern der Open-Model-Community hinterlegt werden, beziehungsweise hat jedes Mitglied bei Bedarf auch einen effizienten Zugriff auf diese Modelle, wodurch die Wiederverwendbarkeit gewährleistet ist. Durch die Entwicklung immer leistungsfähigerer Technologien, welche die modellgestützte Softwareentwicklung vorantreiben, ergibt sich ein immenses Einsparungspotential von Entwicklungszeit und Kapital. Zusätzlich kann diese Modellierungsplattform als

Ausgangspunkt für gemeinschaftliches Arbeiten an Referenzmodellen genutzt werden. Erstellte Modelle könnten kritisch betrachtet und unter den Mitgliedern diskutiert werden. Dadurch würde es zu einem konstruktiven Austausch der Community-Mitglieder kommen, der zu einer weiteren Qualitätssteigerung der angebotenen Modelle führen würde.

## 1.2 Ziel dieser Arbeit

Um die Open-Model-Initiative bei ihren Bemühungen zu unterstützen, eine Community bzw. Plattform zu etablieren, wird in der vorliegenden Arbeit der Problemstellung nachgegangen, wie man Metamodelle in Form einer Web 2.0-Anwendung visualisieren kann. Darüber hinaus wird die Frage behandelt, was für die Visualisierung eines Metamodells erforderlich ist, um einem Anwender die gewünschten Informationen zur Verfügung zu stellen.

Um diese Aufgaben zu lösen, gilt es zuerst zu klären, wie ein Mensch seine Umwelt wahrnimmt und wie er das Wahrgenommene in Form eines Modells und in weiterer Folge in Form eines Metamodells abbilden kann (genau genommen ist die Darstellung eines Metamodells innerhalb einer Web-Applikation die Visualisierung eines **Modells** eines Metamodells). Grundlegend wichtig für die Art der Visualisierung werden die Fragen sein, welche Domäne als vorrangige Quelle für die darzustellenden Metamodelle dient und wer (Adressaten) in erster Linie Informationen aus der Visualisierungs-Applikation (diese Anwendung wird in weiterer Folge als Metamodell-Browser bezeichnet) beziehen kann. Das bedeutet, es muss analysiert werden, welche Kenntnisse bei den Anwendern des Metamodell-Browsers vorauszusetzen sind und wie der Metamodell-Browser den Anwender bei seiner Arbeit unterstützen kann.

Des Weiteren muss geklärt werden, welche Anforderungen an die Web-Applikation zu stellen sind und welche Rich-Internet-Technologie am besten dafür geeignet ist, eine solche Applikation umzusetzen.

Aufbauend auf diese Erkenntnisse kann mit dem eigentlichen Metamodell-Visualisierungsprozess begonnen werden.

## 1.3 Aufbau der Arbeit

Zu Beginn wird in Kapitel 2 (Wissenschaftstheoretische Positionierung) auf den Modellbegriff an sich eingegangen. Darauf aufbauend wird eine Forschungsmethode erarbeitet, in der die wissenschaftstheoretischen Grundannahmen formuliert werden. Die Forschungsmethode hat das Ziel eine "eindeutige" Positionierung zu definieren, die bei den

Lesern dieser Arbeit zu einem einheitlichen Verständnis führen soll. Es werden ein Epistemologischer Bezugsrahmen kreiert und relevante erkenntnistheoretische Fragen diskutiert. Anhand dieser Analysen wird auch ersichtlich werden, mit welchen Schwierigkeiten die **Open-Model-Initiative** bei der Etablierung einer breit aufgestellten Community zu rechnen hat

Nachfolgend werden in Kapitel 3 (Wissenschaftliche Grundlagen) wissenschaftliche Begrifflichkeiten erörtert, die für das Verständnis dieser Arbeit unumgänglich sind, wie z. B.: System, Semiotik, etc.

Das nächste Kapitel 4 (Open Model Initiative) beleuchtet die vorrangigen Adressaten dieser Arbeit. Erst wenn man die Motivation und die Ziele der Open-Model-Initiative nachvollziehen kann, sind auch der Inhalt und der zu realisierende Prototyp dieser Arbeit zu verstehen.

Das darauf folgende Kapitel 5 (Rich Internet Applikationen) beschäftigt sich mit der Wahl einer Rich-Internet-Technologie, mit welcher der Prototyp umgesetzt werden soll. Zu diesem Zweck werden verschiedene Technologien untersucht und bewertet. Jene Technologie, die den relevanten Anforderungen am nächsten kommt, wird in Folge die Grundlage für den Prototyp des Metamodell-Browsers darstellen.

Nach der Wahl einer geeigneten Technologie wird der Metamodell-Browser konzeptionell und prototypisch entwickelt. Kapitel 6 (Konzeptionelle und prototypische Implementierung) wird die Dokumentation des Projektablaufs beinhalten.

Eine Zusammenfassung und ein Ausblick auf mögliche weitere Forschungsziele bilden den Abschluss der Arbeit.

# 2 Wissenschaftstheoretische Positionierung

Dieses Kapitel dient als wissenschaftstheoretische Positionierung für die vorliegende Arbeit. Beginnend mit einer überblicksmäßigen Zusammenfassung des Modellbegriffs im Allgemeinen folgt die Erstellung eines erkenntnistheoretischen Bezugsrahmens. Dieser Bezugsrahmen beeinflusst in weiterer Folge die Wahl der Darstellung des Metamodell-Browsers und dient somit als theoretische Grundlage dieser Arbeit. Insbesondere wird anhand von fünf Aspekten die Problematik analysiert, die eine weitreichende Community bei der Erstellung von Modellen in kooperativer und verteilter Weise mit sich bringt. Bei einem Modell handelt es sich um ein Abbild der Real-Welt und unterliegt somit immer auch subjektiven Einflüssen des Modellerstellers. Da der Metamodell-Browser als Ausgangspunkt für die Entwicklung einer Lösung für ein Modellierungsproblem dienen kann, wird in Folge ein Rahmenwerk definiert, welches für die Mitglieder der Open-Model-Community als Leitfaden für ein einheitliches Modellverständnis sorgen soll.

## 2.1 Der Modellbegriff

Nachdem eine Vielzahl von Definitionen für den Modellbegriff in der Literatur existieren, sei eine der populärsten exemplarisch in Folge angeführt.

**Drei Merkmale des Modellbegriffs nach Herbert Stachowiak:**

**1. Das Abbildungsmerkmal:** *"Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können".*

**2. Das Verkürzungsmerkmal:** *"Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaffern und/ oder Modellbenutzern relevant scheinen".*

Stachowiak versteht unter dem Verkürzungsmerkmal eine Vereinfachung von komplexen Sachverhalten, um die Übersicht zu gewährleisten.

**3. Das pragmatische[1] Merkmal:** *"Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion:*

[1] orientiert sich am praktischen Nutzen des Modells

- *für bestimmte - erkennende und/ oder handelnde, modellbenutzende - Subjekte;*
- *innerhalb bestimmter Zeitintervalle und*
- *unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen".*

[Stac73]

Stachowiak sieht ein Modell nicht zwangsläufig einem bestimmten Original zugeordnet. Es kann für einen bestimmten Personenkreis, für eine bestimmte Zeitspanne oder auch für einen bestimmten Zweck eingesetzt werden.

Bei der Entwicklung des **Metamodell-Browsers** ist insbesondere das pragmatische Merkmal von Bedeutung. Dieses besagt, dass das Verständnis für einen Begriff beim erkennenden Subjekt, z. B.: dem **Chef-Entwickler** oder dem **Anwender** eines Modells bzw. Metamodells, liegt. Daraus lässt sich folgern, dass es keine allgemein gültige und verbindliche Definition für den Modellbegriff geben kann und damit immer ein Interpretationsspielraum bei der Erstellung von (Meta-)Modellen vorhanden ist. Wenn es schon nicht möglich ist, den (Meta)-Modellbegriff eindeutig zu definieren, so ist es zumindest möglich, einen einheitlichen Bezugsrahmen zu erstellen, um den Interpretationsspielraum einzugrenzen.

## 2.2 Erkenntnistheoretischer Bezugsrahmen

In Anlehnung an die von Becker, Niehaves und Knackstedt [vgl. BeckNiehKnack04] formulierten Leitfragen werden in weiterer Folge die erkenntnistheoretischen Grundannahmen formuliert werden, die als Ausgangspunkt für ein einheitliches Modellierungsverständnis für die **Open-Model-Community** dienen sollen und somit auch Einfluss auf die Gestaltung des **Metamodell-Browsers** haben.

Der verwendete Bezugsrahmen bedient sich fünf Aspekten, die für das wissenschaftliche Arbeiten bei der Informationsmodellierung von besonderer Bedeutung sind, wobei sich zu jedem Aspekt mehrere Möglichkeiten der Positionierung ergeben (Abbildung 1).

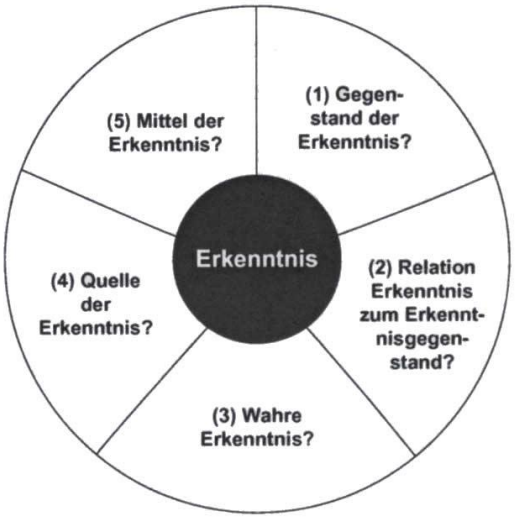


**Abbildung 1: Epistemologischer Bezugsrahmen [BeckNiehKnack04]**

**Punkt 1: Der Gegenstand der Erkenntnis (Ontologischer Aspekt)**

Spricht man von Ontologie, handelt es sich dabei um die Wissenschaft und die Lehre bezüglich des Wesens der Realität, wie wir sie wahrnehmen, beziehungsweise der Existenz im Allgemeinen. Die Ontologie widmet sich somit der Erforschung dessen, "was ist" und "wie es ist". Im Zuge der ontologischen Positionierung soll definiert werden "was" der Gegenstand der wirtschaftswissenschaftlichen Untersuchung darstellt und inwiefern eine Realität als unabhängiger Teil der subjektiven Vorstellung des einzelnen Subjekts existiert. Dabei lassen sich drei Ansichten unterscheiden:

- *Ontologischer Realismus*: Beim ontologischen Realismus besteht die Annahme, dass eine existierende Realität unabhängig von menschlichen Denkprozessen besteht. Die Annahme geht somit davon aus, dass die Realität nicht von einer menschlichen Sprachdefinition abhängt, sondern auch ohne diese existent ist. Folglich handelt es sich bei den Forschungsgegenständen des ontologischen Realismus um Realweltobjekte.
- *Ontologischer Idealismus*: Der ontologische Idealismus geht davon aus, dass die Realität ausschließlich im Zusammenhang mit dem menschlichen Bewusstsein betrachtet werden kann und ein Produkt der menschlichen Wahrnehmung ist. Somit steht die Existenz bzw. die Wirklichkeit im unmittelbaren Bezug zum menschlichen Sprechen und Denken.

- *Kantianismus*: Immanuel Kant vertritt mit seiner Erkenntnistheorie eine offene Position. Auf der einen Seite geht er davon aus, dass Betrachtungsgegenstände auch abseits der menschlichen Wahrnehmung existieren ("Dinge an sich"), aber auch Objekte existieren, die an ein menschliches Bewusstsein gebunden sind (Regulative Ideen (Seele, Freiheit, Gott)).

Da es sich bei den Forschungsgegenständen dieser Arbeit um (Referenz-)Modelle und die diese (Referenz-)Modelle beschreibenden (Referenz-)Metamodelle handelt, erscheint es sinnvoll, den Standpunkt des ontologischen Realismus einzunehmen. Modelle (z. B. der Wirtschaftsinformatik) dienen in erster Linie als Abbildungen realer Zustände (bzw. geplanter realer Zustände) unabhängig vom menschlichen Bewusstsein und auch unabhängig von Denk- und Sprechprozessen. Die Erstellung von (Referenz-)Metamodellen und in weiterer Folge die **Visualisierung von Metamodellen** kann man somit als real existierendes Problem bezeichnen, das gelöst werden soll.

**Punkt 2: Relation der Erkenntnis zum Erkenntnisgegenstand (Epistemologischer Aspekt)**

Eine zentrale Frage der Erkenntnistheorie ist das Verhältnis der durch ein Subjekt (z. B.: **Chef-Entwickler der Open-Model-Initiative**) erworbenen Erkenntnis gegenüber dem Erkenntnisgegenstand (z. B.: **Darstellung eines Metamodells im Metamodell-Browser**), wobei es sich beim Erkenntnisgegenstand um ein Objekt handelt, das einem Subjekt bzw. Erkennenden eine Erkenntnis bringen soll. Dass es dabei unweigerlich zu Differenzen kommen muss, resultiert aus der Verschiedenheit der Erkennenden (sei es aufgrund unterschiedlicher Ausbildung, unterschiedlicher Intelligenz, unterschiedlicher Standpunkte[2] usw.). Prinzipiell wird untersucht, inwieweit der Mensch in der Lage ist, Dinge objektiv zu erkennen, die unabhängig vom menschlichen Bewusstsein existieren. Dafür gibt es zwei divergente Ansichten:

- Der *Konstruktivismus* vertritt die Ansicht, dass Erkenntnis immer subjektiv ist und die Relation zwischen Erkenntnis und Erkenntnisgegenstand immer vom erkennenden Subjekt bestimmt wird.

[2] Beispiel: Bei der Planung für eine Welle im Maschinenbaubereich wird ein Ingenieur in erster Linie die Funktionalität der Welle berücksichtigen. Für den Dreher, der die Welle anzufertigen hat, ist in erster Linie entscheidend, ob er die Welle in die Drehbank einspannen kann, um sie plankonform zu fertigen.

- Der *erkenntnistheoretische Realismus* hingegen schließt nicht aus, dass eine objektive Wahrnehmung einer unabhängigen Real-Welt möglich ist. Durch die Setzung geeigneter Maßnahmen könnten Störgrößen, die zu subjektabhängigen Verzerrungen führen, kompensiert werden, wodurch die Erkenntnis für alle betrachtenden Subjekte gleich wäre.

Verfolgt man den Ansatz des erkenntnistheoretischen Realismus und der Möglichkeit einer objektiven Erkenntnis, so impliziert dies, dass auch eine vom menschlichen Bewusstsein unabhängige, objektive Real-Welt existieren muss (siehe ontologischer Realismus). Die konstruktivistische Sichtweise bezüglich Erkenntnis und Erkenntnisgegenstand schließt hingegen keine der zuvor erläuterten ontologischen Sichtweisen aus. Abbildung 2 zeigt die Ausprägungsmöglichkeiten, die sich durch die Kombination der erkenntnistheoretischen und ontologischen Fragestellungen ergeben können.

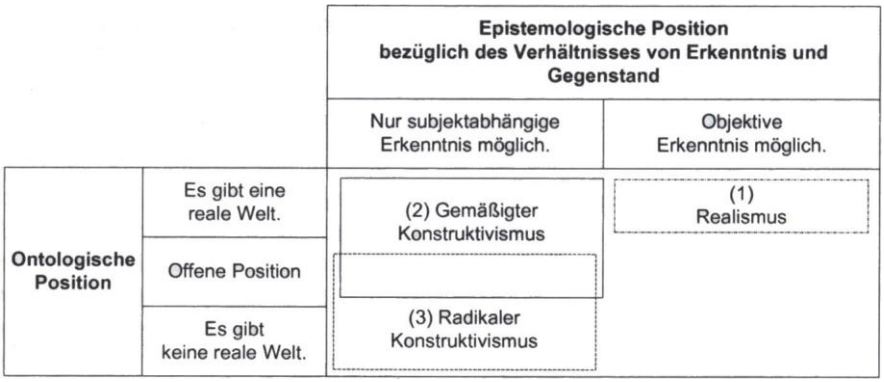


**Abbildung 2: Kombinationsmöglichkeiten ontologischer und epistemologischer Positionen [BeckNiehKnack04]**

(1) Der *Realismus* geht davon aus, dass es eine objektive Real-Welt gibt, die unabhängig vom menschlichen Bewusstsein existiert und auch als solche objektiv für jeden Menschen erkennbar ist.

(2) Angelehnt an den Kantianismus verhält es sich beim *gemäßigten Konstruktivismus*. Dabei wird von einer existierenden Real-Welt ausgegangen, jedoch dem subjektiven Erkenntnisprozess des einzelnen Individuums große Bedeutung zugesprochen. Die Subjektivität ergibt sich hauptsächlich durch den Versuch, die Realität durch die menschliche Sprache nach- bzw. abzubilden. Das hat zur Folge, dass die verwendete Sprache oft nur innerhalb einer bestimmten Domäne (z.B.: Ärzte) bzw. Sprachgemeinschaft einer einheitlichen Interpretation unterliegt und somit ein und dieselbe Beschreibung in zwei

unterschiedlichen Domänen ein sehr unterschiedliches Abbild eines vermeintlich gleichen objektiven Sachverhaltes zur Folge haben kann.

(3) Beim *radikalen Konstruktivismus* wird die Sicht auf eine mögliche objektive Real-Welt zur Gänze negiert und somit wäre jeder Erkenntnisprozess für jedes Subjekt individuell. Daraus könnte man schließen, dass sich jede persönliche Erkenntnis ausschließlich auf die eigene subjektive Wirklichkeit beziehen kann und somit jedes Subjekt einer eigenen abgeschlossenen Domäne entsprechen würde.

Im Zuge dieser Arbeit wird in weiterer Folge ein gemäßigt konstruktivistischer Ansatz verfolgt. Die **Visualisierung von Metamodellen** und deren Interpretation setzten Kriterien voraus, ohne die ein unbedarfter Betrachter nicht in der Lage sein dürfte, auch nur die kleinste Erkenntnis aus den Ergebnissen zu beziehen. Somit ist es wahrscheinlich, dass die Zielsubjekte, für die der **Metamodellbrowser** gedacht ist, aus sich überschneidenden Domänen entstammen und dadurch eine prinzipiell gleiche bzw. sehr ähnliche Interpretation der **Metamodelle** erfolgen wird. Eine realistische Sichtweise erscheint hingegen nicht zielführend, weil ansonsten sämtliche betrachtenden Subjekte ein und derselben Domäne entstammen müssten. Das ist erstens äußerst "unrealistisch" und zweitens noch immer keine Garantie dafür, dass eine völlig objektive Erkenntnis vorliegt.

**Punkt 3: Wahre Erkenntnis (Der Wahrheitsbegriff)**

Der **Metamodellbrowser** soll es der **Open-Model-Initiative bzw. der Community** ermöglichen, aus den dargestellten Metamodellen "wahre Erkenntnis" über die zugrunde liegenden Metamodelle zu erlangen und sie sollten in der Lage sein, "richtiges" Wissen aus den Darstellungen zu erwerben. Es existieren drei gängige Theorien zum Wahrheitsbegriff.

- *Korrespondenztheorie der Wahrheit*: Bei der Korrespondenztheorie werden zwei Begriffe einander gegenüber gestellt. Auf der einen Seite die Aussage und auf der anderen Seite die Tatsache, mit deren Hilfe der Wahrheitsgehalt einer Aussage überprüft werden kann. Durch Tatsachen kann somit festgestellt werden, ob Aussagen wahr oder falsch sind, wobei man davon ausgehen muss, dass Tatsachen prinzipiell objektiv wahr sind. Diese Theorie setzt eine sowohl erkenntnis-theoretisch als auch ontologisch realistische Positionierung voraus, weil davon ausgegangen wird, dass es eine objektive Wahrheit gibt und dadurch Tatsachen als realweltliche Objekte betrachtet werden müssen.

- *Konsenstheorie der Wahrheit*: Bei der Konsenstheorie wird nicht von einer für alle gültigen objektiven Real-Welt ausgegangen. Vielmehr folgt die Konsenstheorie einem konstruktivistischen Ansatz. Demzufolge gilt eine Aussage dann als "wahr", wenn die Aussage unter den Subjekten innerhalb einer bestimmten Gruppe bzw. Domain als "wahr" akzeptiert wird. Somit sind eine Aussage und das dazugehörende Wahrheitsverständnis immer in Relation zu einer bestimmten Gruppe zu sehen.

  Zum Beispiel die Aussage: *"Hansi Hinterseer macht schöne Musik"* wird bei Liebhabern der volkstümlichen Musik zu einem anderen Wahrheitsverständnis führen als bei den meisten anderen Domänen auf dieser Welt.

  Die Tatsache, dass sich die Wahrheit als Konsens einer bestimmten Gruppe manifestiert, macht es aber unmöglich, die Aussagen auf ihre Richtigkeit zu prüfen.

- *Semantische Theorie der Wahrheit*: Bei dieser von TARSKI geprägten Wahrheitstheorie wird im Gegensatz zur Korrespondenztheorie der Wahrheitsgehalt einer Aussage nicht gegen allgemeingültige objektive Tatsachen geprüft, sondern gegen eine zuvor genau definierte **Objektsprache**. Das bedeutet, dass eine Aussage immer dann als "wahr" zu bezeichnen ist, wenn sie im Sinne der Objektsprache wahr ist. Demzufolge ist der Wahrheitsgehalt solcher Aussagen immer nur relativ zu der dazugehörigen Objektsprache zu verstehen. Die Definition der Objektsprache erfolgt durch die Formulierung in einer, für genau diese Objektsprache geltenden, **Meta-Sprache.**

  In Bezug auf das vorige Beispiel bedeutet das: Würde in einer Metasprache definiert sein: *"Die Musik von Hansi Hinterseer ist der letzte Bockmist"* könnte bei Nutzern der dazugehörenden Objektsprache der weiter oben formulierte Beispielsatz nie eine "wahre" Aussage ergeben.

  Durch dieses Vorgehen kann eine Konsensbildung in einer Gruppe von Individuen erreicht werden, obwohl die einzelnen Subjekte aus verschiedenen Domänen stammen, die prinzipiell nicht die gleiche "Sprache" sprechen. Voraussetzung für die Nutzung einer bestimmten Objektsprache für eine Konsensgemeinschaft ist die

Verfügbarkeit der dazu gehörenden Metasprache, anhand derer der Wahrheitsgehalt der formulierten Aussagen geprüft werden kann.

Da es sich bei den Adressaten dieser Arbeit um Angehörige der **Open-Model-Community** handelt, ist sowohl die Konsenstheorie als auch die Semantische Theorie von Relevanz. Man kann davon ausgehen, dass ein breiter Konsens bei der Beurteilung der Aussagen, ihren Wahrheitsgehalt betreffend, existiert, da die Gruppe ein sehr ähnliches Verständnis über z. B. Wirtschaftsinformationsmodelle besitzen sollte. Die Bedeutung der semantischen Theorie liegt bei der vorliegenden Arbeit im Thema an sich. Für die Benutzer des **Metamodell-Browsers** werden anhand der Metamodelle die dazugehörigen Objektsprachen definiert, mit deren Hilfe Modelle für den z. B. Wirtschaftsinformationsbereich erstellt werden können. Sollte man in Zukunft der **Open-Model-Community** die Möglichkeit bieten wollen, zu weiteren Forschungszwecken auch die Metamodelle für neue Aufgabengebiete adaptieren und erweitern zu können, müssten der Community auch die **Metametasprachen**, welche die Metamodelle beschreiben, zur Verfügung gestellt werden.

**Punkt 4: Quelle der Erkenntnis (Woher stammt Erkenntnis?)**

Der Ursprung unseres Wissens und die Entstehung von Erkenntnisinhalten, auf denen unser Wissen beruht, sind direkt auf das menschliche Erkenntnisvermögen zurückzuführen. Wie schon bei den vorherigen Aspekten, lassen sich bei der Quelle der Erkenntnis unterschiedliche Standpunkte einnehmen, wie die Erkenntnisinhalte vermittelt werden können. Speziell beim Versuch, **Metamodelle visuell** zu vermitteln, sollte auf diesen Sachverhalt Rücksicht genommen werden.

- Der *Empirismus* nimmt an, dass aus Erfahrungen Wissen generiert werden kann. Das empirische bzw. aposteriorische Wissen wird nicht aufgrund intellektueller Fähigkeiten erlangt, sondern folgt eher einem Versuch-Irrtum-Prinzip, das sich mehr auf eine naturwissenschaftliche Theorie- und Erfahrungspraxis stützt.
- Nimmt man den menschlichen Verstand bzw. Intellekt als Wissensquelle an, spricht man vom *Rationalismus*. Beim Rationalismus wird davon ausgegangen, dass ein Subjekt mit Hilfe seiner intellektuellen Fähigkeiten Objekte aufgrund deren Eigenschaften unterscheiden und durch gedankliche Reflexion neue Erkenntnisse erschließen kann.
- Der *Kantianismus* bedient sich wieder eines Mittelwegs der beiden zuvor beschriebenen Positionierungen. Es wird sowohl der Verstand als auch die Erfahrung

eines Subjekts als Quelle des Wissens berücksichtigt. Kant geht dabei sogar einen Schritt weiter und vertritt die Meinung, dass *nur* die Kombination aus empirischen und rationalen Quellen zu einem Erkenntnisgewinn führen kann.

Im Zusammenhang mit dieser Arbeit erscheint eine kantianistische Positionierung sinnvoll. Auf der einen Seite muss die praktische **Handhabbarkeit des Metamodell-Browsers** empirisch betrachtet werden. Andererseits sollten die **visualisierten Metamodelle** auf einer hohen intellektuellen Abstraktionsebene den interessierten Anwendern Erkenntnisse über die einzelnen Komponenten und deren Abhängigkeiten bieten. Zu diesem Zweck wird im Anschluss an die konzeptionelle Erstellung des Metamodelbrowsers eine prototypische Implementierung folgen.

**Punkt 5: Mittel der Erkenntnis (Methodologischer Aspekt)**

Beim Methodologischen Aspekt versucht man zu definieren, wie man neues Wissen herleiten kann. Folgende drei Vorgehensweisen sind im Zusammenhang mit der **Darstellung von (Referenz-)Metamodellen** relevant:

- Der *Induktivismus* versucht durch Induktion, also dem Schließen von einem speziellen Einzelfall zu einer allgemeingültigeren Aussage, zu neuer Erkenntnis zu gelangen. Es wird aus einem Einzelfall eine Verallgemeinerung abgeleitet. Dieses Vorgehen führt oft zu einer Generalisierung und findet seine Anwendung hauptsächlich nach empirischen Beobachtungen im naturwissenschaftlichen Bereich.
- Der *Deduktivismus* geht einen umgekehrten Weg. Unter Verwendung von logischen Schlussregeln wird versucht, aus bestehenden Aussagen eine neue Aussage zu formulieren. Es wird also aus dem Allgemeinen ein Einzelfall abgeleitet. Ein Beispiel dafür ist ein Logical: Durch mehrere Hinweise (Schlussregeln) wird es möglich eine neue Fragestellung (Aussage) zu lösen.
- Die *Hermeneutik* beschäftigt sich mit dem Menschen als verstehendes Subjekt. Grundvoraussetzung für das Verstehen eines Sachverhalts ist ein einschlägiges Vorwissen. Sofern ein Mensch über das nötige Vorwissen und einen entsprechenden Intellekt verfügt, sollte er in der Lage sein, einen Sachverhalt zu analysieren und neu zu interpretieren. Als Folge der Neuinterpretation kann das eigene Wissen an den neuen Sachverhalt angepasst werden und somit neue Erkenntnisse erlangt werden. Aufkommende Fragen können zu weiteren Fragen und Diskussionen eines

Sachverhalts führen. Dieser Kreislauf wird auch als hermeneutischer Zirkel bezeichnet.

Bei der Konzeptionierung und Anwendung des **Metamodell-Browsers** wird die induktive und deduktive Vorgehensweise zur Erlangung neuer Erkenntnis nur eine untergeordnete Rolle spielen. Im Gegensatz dazu wird die Hermeneutik von großer Bedeutung sein, bedenkt man, dass die **Anwender des Browsers** in der Lage sein müssen, Sachverhalte der **Domäne Metamodellierung** interpretieren zu können. Dazu muss man domänenspezifisches Vorwissen voraussetzen können. Dieses Vorwissen sollte bei der **Open-Model-Community** als Adressat zu einem großen Teil vorhanden sein.

# 3 Wissenschaftliche Grundlagen

Dieses Kapitel erläutert jene Themengebiete und deren Zusammenhänge, die für den Einsatz des **Metamodell-Browsers** im Sinne der **Open-Model-Initiative** von Relevanz sind. Durch die selbstauferlegte Beschränkung der **Open-Model-Initiative,** vorerst nur Modelltypen zu unterstützen, die ein schnelles und intuitives Verstehen ermöglichen (siehe Kapitel 4.3), soll auch die Gestaltung des Metamodell-Browsers diesem Motto (schnelles und intuitives Verstehen) entsprechen. Die unterstützten Modelltypen (Datenmodelle, Prozessmodelle und Wissensmodelle) können unter dem allgemeineren Begriff „**Informationsmodelle“** zusammengefasst werden. Diese sind die abstrakte Abbildung von **Informationssystemen**. Im Anschluss werden die wichtigsten Begriffe von Informationssystemen erörtert und aufgezeigt, an welcher Stelle der Metamodell-Browser bei der Erstellung eines Informationsmodells zu positionieren ist.

## 3.1 System

Ein System beschreibt ein Gebilde, das Elemente, Beziehungen zwischen den Elementen und Beziehungen zwischen den Eigenschaften der Elemente beinhaltet und sich in seiner Gesamtheit durch seine Gliederung und Ordnung von der restlichen Umwelt abgrenzt [vgl. HALLFAGEN1956]. Als restliche Umwelt bezeichnet man jene Elemente, die sich außerhalb des betrachteten Systems befinden.

Bei Systemen können gedankliche und natürliche Systeme unterschieden werden. Dabei sollte man differenzieren, welche erkenntnistheoretische Positionierung vertreten wird. Natürliche Systeme implizieren eine zugrundeliegende Real-Welt, von der ein natürliches System abgeleitet werden kann, und setzt den Standpunkt des Realismus voraus. In Bezug auf diese Arbeit wird aber der erkenntnistheoretische Standpunkt des gemäßigten Konstruktivismus eingenommen, demzufolge davon ausgegangen wird, dass ein System immer in direktem Zusammenhang mit einer gedanklichen Konstruktionsleistung eines Subjekts verbunden ist.

Inwieweit ein System als offen oder geschlossen (Abbildung 3) angesehen wird, hängt davon ab, ob Elemente innerhalb des Systems Beziehungen zu Elementen außerhalb des Systems eingehen oder nicht. Ob ein Element einem System zugehörig ist oder nicht, unterliegt der Empfindung des Systemkonstrukteurs [vgl. DELFMANN2006].

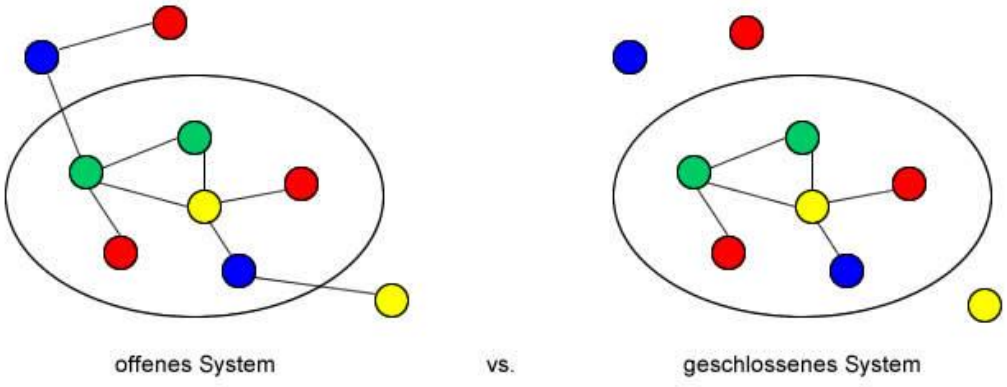


**Abbildung 3: Offenes vs. geschlossenes System [vgl. HILLFEHLUL1989]**

Des Weiteren lassen sich drei unterschiedliche Aspekte bei Systemen unterscheiden [vgl. DELFMANN2006]:

- Strukturaler Aspekt: Der strukturale Aspekt legt das Hauptaugenmerk auf die im System befindlichen Elemente und deren Beziehungen untereinander. Es wird also rein die Struktur des Systems betrachtet. Bleibt der Zustand eines Systems über die Zeit betrachtet immer gleich, spricht man von einem statischen System, zum Beispiel ein ohmscher Widerstand (egal welche Spannung anliegt, der Widerstand bleibt immer gleich). Dynamische Systeme können ihren Zustand über die Zeit gesehen verändern, zum Beispiel ein Regelkreis.
- Funktionaler Aspekt: Bei dynamischen Systemen lässt sich ein Systemverhalten erkennen, welches als funktionaler Aspekt bezeichnet wird. Offene dynamische Systeme können ihr Systemverhalten durch Inputparameter, die das System von außen erhält, verändern und dadurch Outputs generieren, die wiederum die Umwelt außerhalb des Systems beeinflussen. Die verschiedenen Systemzustände, die durch die Eingabe verschiedener Inputwerte entstehen können, ergeben sich aus der Struktur des Systems (siehe struktureller Aspekt).
- Hierarchischer Aspekt: Oft besteht ein größeres System nicht nur aus einzelnen Elementen sondern auch aus mehreren kleineren Systemen. Diese kleineren Systeme werden als Subsysteme bezeichnet, wobei diese wiederum weitere Subsysteme beinhalten können. Ist ein Subsystem Teil eines übergeordneten Systems, wird das übergeordnete System als Supersystem bezeichnet. Somit kann jedes Systemelement, welches sich noch nicht auf der hierarchisch niedrigsten Ebene befindet, in weitere Systemelemente untergliedert werden, womit die Komplexität eines großen Systems

überschaubarer wird. Im Zusammenhang mit dieser Arbeit stellt die Modellierung von Geschäftsprozessen einer Firma ein gutes Beispiel dar. Würde man die Geschäftsprozesse nicht in einzelne Subprozesse zerlegen, wäre das Informationsmodell der Firma extrem groß und somit nicht mehr überschaubar. Ähnliches gilt für die Darstellung von Metamodellen im **Metamodell-Browser**. Bei großen Metamodellen muss die Möglichkeit geschaffen werden, nur Teilbereiche zu betrachten bzw. nur Teilbereiche heraus zu zoomen (z. B. mit einer Fischauge-Funktion).

## 3.2 Semiotik

Als Semiotik wird die Lehre von Zeichen, Zeichensystemen und Zeichenprozessen beschrieben und ist Teil der philosophischen Erkenntnistheorie. Unter einem semiotischen Vorgang in sprachwissenschaftlicher Hinsicht versteht man die Übermittlung einer codierten Nachricht von einer sendenden Einheit an eine empfangende Einheit, welche die Nachricht decodieren bzw. entschlüsseln kann. Auf diese Weise entsteht eine Interaktion zwischen Sender und Empfänger [vgl. LORENZ1995].

Die Arten der Nachrichten bewegen sich auf unterschiedlichen Ebenen der Semiotik. Je nachdem auf welcher Ebene der Nachrichtenaustausch vollzogen wird, kann man den Charakter der Nachrichten als Daten, Informationen und Wissen bezeichnen [vgl. DELFMANN2006]:

- **Daten** befinden sich auf der syntaktischen Ebene. Die Syntax bzw. Grammatik der Daten ist korrekt, wenn die verwendeten Zeichen aus einem gegebenen Zeichenvorrat stammen und die Zeichenfolge bestimmten Mustern entspricht. Ein solcher Zeichenvorrat könnte zum Beispiel das lateinische Alphabet sein und ein Muster für eine korrekte Zeichenfolge die deutsche Sprache. Bezogen auf eine natürliche Sprache ist somit nur die korrekte Aneinanderreihung der einzelnen Buchstaben innerhalb der einzelnen Wörter bzw. der Wörter eines Satzes ausschlaggebend. Die Bedeutung der einzelnen Wörter bzw. ob die Wörter innerhalb eines Satzes einen Sinn ergeben, ist syntaktisch nicht relevant. Das bedeutet in weiterer Folge, dass Daten zwar korrekte Zeichenfolgen sind, aber keine weitere Bedeutung haben und auch keiner weiteren Interpretation bedürfen. Durch diese rudimentären Eigenschaften sind Daten sehr gut für einen vollautomatischen Nachrichtenaustausch geeignet und können leicht auf ihre syntaktische Korrektheit überprüft werden.

- Sobald Daten eine bestimmte Bedeutung zugewiesen wird, begibt man sich von der syntaktischen Ebene auf die semantische Ebene. Aus den bedeutungslosen Daten werden somit **Informationen**. Um eine Bedeutung aus Daten extrahieren zu können, ist in weiterer Folge die korrekte Interpretation eines Nachrichtenempfängers erforderlich. Dieser muss, dem hermeneutischen Prinzip folgend, das gleiche Vorwissen haben wie der Nachrichtensender. Bei menschlichen Individuen setzt dies in erster Linie den Gebrauch der gleichen Sprache und zusätzlich eine sich überschneidende Vorbildung voraus, um den semantischen Zusammenhang zu erkennen. Soll der Informationsaustausch automatisiert durchgeführt werden (Maschine zu Maschine), so ist die Interpretation durch beschreibende Metadaten möglich, deren Definition für beide Maschinen gleich sein muss[3].
- Um aus Informationen **Wissen** zu erlangen, müssen neue Informationen durch ein Individuum aufgenommen und danach bewertet werden. Dieses Handeln findet auf der pragmatischen Ebene statt und entspricht einem Lernprozess, in dem die Informationen dem Erfahrungsschatz des Individuums hinzugefügt werden. Durch das Bewerten können Informationen als "wahr" oder "falsch" eingestuft werden und haben daraufhin unmittelbaren Einfluss auf zukünftige Entscheidungen eines Subjekts.

In der **Wirtschaftsinformatik** ist es jedoch sehr wichtig, auch den umgekehrten Weg zu betrachten. Wie die Bezeichnung "Wirtschaftsinformatik" schon andeutet, sollen Informationen für betriebswirtschaftliche Tätigkeiten bereitgestellt werden. Dazu muss zuerst das vorhandene Wissen in Informationen und anschließend in Daten abstrahiert werden. Die Abstraktion von Wissen kann durch die Bereitstellung von Zeichenketten oder (wie im Fall des **Metamodell-Browsers**) durch graphische Symbole erfolgen, die durch Individuen interpretiert werden können, sofern sie in einer syntaktisch korrekten Form vorliegen. Um die Informationen über z. B. das Internet zu verbreiten, müssen diese in eine maschinenverständliche Form, also in Daten, transformiert werden.

[3] Man sollte dabei jedoch bedenken, dass die semantischen Beschreibungen in Metadaten aus Sicht der Maschine ebenfalls nur Zeichenketten sind, welche die Maschine abarbeitet. Somit bleibt es der Interpretation des Betrachters überlassen, inwieweit eine Maschine "wirklich" Informationen verarbeiten kann.

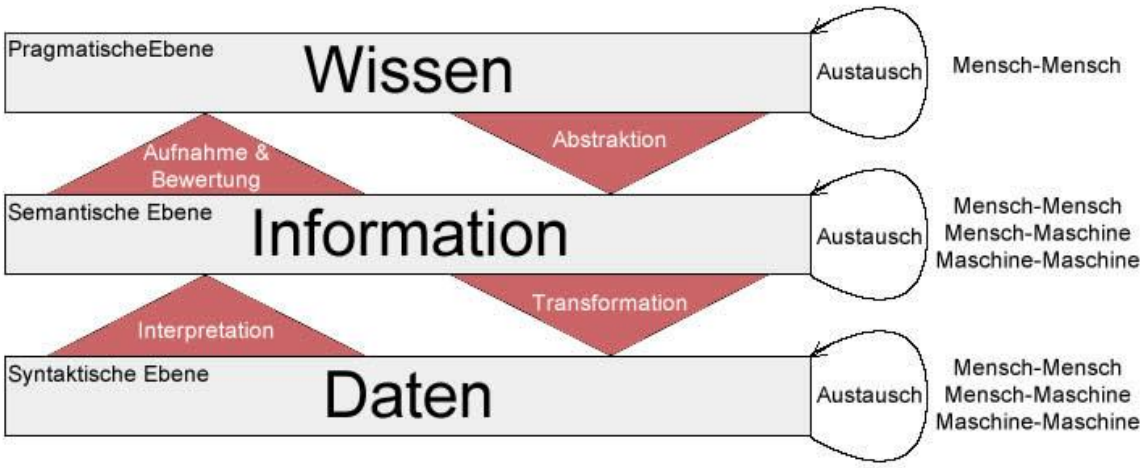


**Abbildung 4: Zusammenhang und Austausch von Wissen, Informationen und Daten [vgl. DELFMANN2006]**

In Abbildung 4 ist der Zusammenhang der semiotischen Ebenen dargestellt und wie Daten in Informationen bzw. Informationen in Wissen überführt werden können und umgekehrt. Ebenso sind die Instanzen ersichtlich, zwischen denen Daten, Informationen und Wissen ausgetauscht werden können. Bezugnehmend auf die Auszeichnungssprache XML kann bspw. ein Informationsaustausch direkt zwischen Mensch und Maschine auf der semantischen Ebene durchgeführt werden[4].

## 3.3 Informationssysteme

Um Informationen verarbeiten zu können, werden spezielle Systeme benötigt, s. g. Informationssysteme. Anhand der zuvor erörterten Begriffe "System" und "Information", die im Zusammenhang mit dieser Arbeit von beträchtlicher Relevanz sind, lassen sich die folgenden Eigenschaften für Informationssysteme herleiten.

Um Informationen in Firmen bzw. Unternehmen verarbeiten zu können, sind verschiedene Elemente innerhalb eines Informationssystems erforderlich. Wie schon zuvor erwähnt, sind es die Individuen bzw. Mitarbeiter innerhalb eines Unternehmens, welche die zur Verfügung gestellten Informationen interpretieren sollen. Ein Subsystem bzw. Element innerhalb eines Informationssystems ist das **Organisationssystem**. Das Organisationssystem dient dazu, die fachlichen und disziplinären Abhängigkeiten der einzelnen Individuen innerhalb eines Unternehmens festzulegen und ist somit unerlässlich für ein Informationssystem.

[4] Eine positive Eigenschaft von XML ist deren Syntax, die sowohl für Menschen als auch von Maschinen semantisch interpretierbar ist.

Prinzipiell sind Individuen auch ohne technische Hilfsmittel in der Lage, bereitgestellte Daten als Information zu interpretieren. Es ist aber anzunehmen, dass in einem Unternehmen Daten in sehr großer Menge anfallen können und diverse Daten erst in Kombination als s. g. Kennzahlen die gewünschten Informationen bringen. Um diese Datenflut bewältigen zu können und um Daten in einem annehmbaren Zeitrahmen als Information zur Verfügung gestellt zu bekommen, scheint es zweckmäßig, die anfallenden Datenmengen automatisiert zu verarbeiten. Diese maschinenunterstützte Aufbereitung der Daten findet in einem **Datenverarbeitungssystem** statt, welches somit ein weiteres Element innerhalb eines Informationssystems darstellt. Die Umsetzung der maschinenunterstützten Datenverarbeitung findet durch Computersysteme statt.

Das Datenverarbeitungssystem besteht aus mehreren Subsystemen. Im Wesentlichen sind dies das Hardware- und das Softwaresystem. Die Hardware umfasst sämtliche Komponenten eines Computers einschließlich der Peripheriegeräte (z. B.: Scanner, Webcam, ...). Das Softwaresystem setzt sich aus dem Betriebssystem und diverser Anwendungssoftware zusammen.

Organisationssystem und Datenverarbeitungssystem sind somit die wichtigsten Elemente bzw. Subsysteme eines Informationssystems (Abbildung 5).

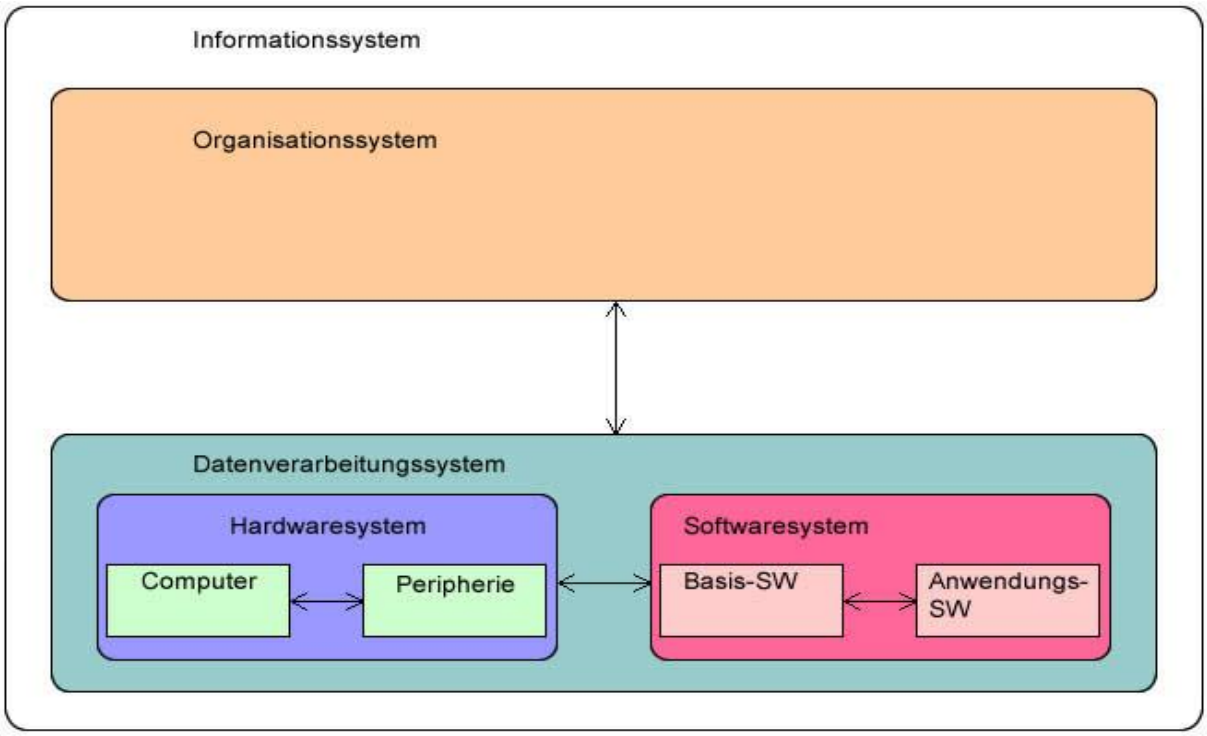


**Abbildung 5: Informationssystem [in Anlehnung an TEUBNER1999]**

Durch Betrachtung des Strukturaspekts, des Verhaltensaspekts und des hierarchischen Aspekts lässt sich der Systemcharakter eines Informationssystems erklären. Die Struktur des Informationssystems (Abbildung 5) wird durch die Untergliederung in einzelne Subsysteme und deren Beziehungen zueinander ersichtlich, zum Beispiel die Kommunikation eines Individuums des Organisationssystems mit einer Benutzerschnittstelle des Datenverarbeitungssystems. Die Verarbeitung von Daten für die Abwicklung von Geschäftsprozessen in einem Informationssystem repräsentiert den funktionalen Aspekt (Daten werden in das System eingegeben - Die Daten werden verarbeitet - Es wird ein Ergebnis generiert). Der hierarchische Aspekt wird durch die Gliederung des Informationssystems in Subsysteme, die wiederum Subsysteme beinhalten, deutlich.

Um die für ein Unternehmen relevanten Informationen zu erhalten, müssen Informationssysteme über Schnittstellen zur Außenwelt verfügen, wodurch sie immer **offene Systeme** sein müssen. Dadurch, dass die über die Schnittstellen erhaltenen Informationen in weiterer Folge verarbeitet werden, erfüllen sie auch die Eigenschaft von **dynamischen Systemen**.

Der Gradmesser für ein Informationssystem ist die Qualität der verfügbaren Informationen, wobei sich die Qualität anhand der folgenden Faktoren beurteilen lässt [vgl. LASSMANN2006]:

- Aktualität;
- Vollständigkeit;
- Verfügbarkeit;
- Genauigkeit.

Damit ein Informationssystem in einem Unternehmen erfolgreich ist, müssen diese Faktoren in ausreichendem Umfang erfüllt sein. Damit spielt die Gestaltung von Informationssystemen eine sehr große Rolle, weil ein funktionstüchtiges Informationssystem für ein Unternehmen einen entscheidenden Wettbewerbsfaktor darstellt. Je höher die Qualität der verfügbaren Informationen ist, desto schneller, genauer und erfolgsversprechender können Entscheidungen innerhalb eines Unternehmens getroffen werden. Die Gegenstandsbereiche, welche durch ein Informationssystem unterstützt werden sollen, können mitunter sehr umfangreich sein. Deshalb erscheint es sinnvoll, den Informationssystemgestaltern die Erfassung der komplexen Gegenstandsbereiche durch Abstraktion in Form von **Informationsmodellen** zu erleichtern.

## 3.4 Informationsmodelle

Ein Informationsmodell dient der abstrakten Repräsentation eines bestimmten Sachverhalts. Einzelnen Subjekten soll die Erfassung der Komplexität von Sachverhalten somit erleichtert werden. Berücksichtigt man die konstruktivistische Positionierung dieser Arbeit, ist ein Modell das Ergebnis eines konstruierenden Subjekts, mit dem Ziel, einen Sachverhalt bzw. ein Original für eine bestimmte Adressatengruppe, unter Nutzung einer bestimmten Sprache, begreifbar zu machen.

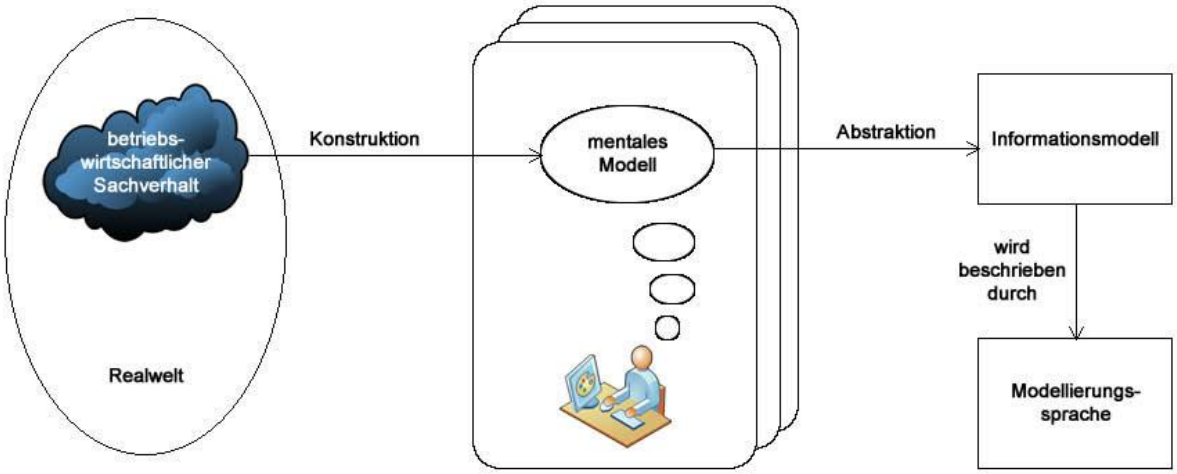


**Abbildung 6: Modellerstellung [vgl. BECKSCH2004]**

Die Modellerstellung an sich unterliegt immer der schöpferischen Fähigkeit eines Subjekts und ist im direkten Zusammenhang mit der individuellen Vorstellungskraft des Subjekts zu betrachten. Ein Subjekt versucht einen Sachverhalt zu repräsentieren, indem es seine Eindrücke, einer als existent angenommenen Real-Welt, in die Erstellung eines Modells einfließen lässt. Somit ist der Ausgangspunkt für die Erstellung eines Modells immer ein individuelles **mentales Modell** [vgl. BECKSCH2004].

Die Erstellung von Informationsmodellen ist nicht zwangsläufig das konstruktive Ergebnis nur eines Subjekts bzw. Modellierers. Je komplexer der zu modellierende Sachverhalt ist, desto eher werden die Modelle in kooperativer Zusammenarbeit von mehreren Modellierern (z. B.: **Entwickler der Open-Model-Community**) durchgeführt werden. Um einen breiten Konsens bei der Erstellung von Modellen zu gewährleisten, ist die Einbeziehung der potenziellen Adressaten bzw. Modellnutzer bei einer kooperativen Modellierung sinnvoll. Damit können etwaige Unstimmigkeiten, das Wahrheitsverständnis betreffend, schon zu

einem sehr frühen Entwicklungsstadium ausgeräumt werden und somit zeit- und kostenintensive Nachbesserungen vermieden werden. Auf die Bedeutung der kooperativen Modellierung im Sinne der Open-Model-Initiative wird noch zu einem späteren Zeitpunkt eingegangen.

Auf welche Art ein Modell einen Sachverhalt repräsentiert, ist abhängig von der verwendeten Modellierungssprache. Die konkrete Syntax bzw. die Darstellungsform einer Modellierungssprache kann graphisch, d. h. in Form von Diagrammen, oder textuell sein [vgl. ENGELS2008].

Ein Modell repräsentiert ein Original so wie es der Modellierer in seiner Vorstellung der Real-Welt wahrnimmt. Der modellierte Sachverhalt ist ein beliebiger, bspw. ein Bauplan eines Hauses, ein Geschäftsprozess in einer Firma oder eine Organisationsstruktur in einem Unternehmen. **Ist-Modelle** repräsentieren die Konstruktion bereits existenter Sachverhalte aus der Real-Welt. Von **Soll-Modellen** spricht man, wenn das Modell mit dem Ziel konstruiert wird, Neues zu gestalten, d. h. das Original existiert zum Zeitpunkt der Modellierung noch nicht. Daher kann es durchaus passieren, und das passiert sehr oft, dass es Soll-Modelle von Sachverhalten gibt, deren originales Pendent in der Real-Welt nie umgesetzt wird [ vgl. SCHÜ1998].

Modelle stehen mit ihren Originalen immer in einem zeitlichen Bezug. Dieser beginnt mit der Erstellung bzw. dem Konstruktionszeitpunkt des Modells und erstreckt sich über die Dauer, in der das Modell seine Gültigkeit hat, d. h. solange das Modell im Bezug zum Original relevant ist. Ändert sich durch äußere Einflüsse das Original, bspw. die Organisationsstruktur in einem Unternehmen, muss das Modell entsprechend angepasst werden oder es verliert seine Gültigkeit [vgl. BECKSCH2004].

Zur Gestaltung von Informationssystemen werden Informationsmodelle verwendet. Der allgemeinen Definition von Modellen folgend, dienen Informationsmodelle den Gestaltern von Datenverarbeitungs- und Organisationssystemen der domänenspezifischen Abstraktion von betriebswirtschaftlichen Sachverhalten. Wie jedes Modell sind Informationsmodelle das konstruktive Ergebnis eines bzw. mehrerer Subjekte und beschreiben die Elemente eines Informationssystems in einem relevanten Zeitrahmen unter Verwendung einer Modellsprache.

Bei der Gestaltung von Informationssystemen hat sich eine iterative d. h. schrittweise Entwicklung als tauglichste Vorgehensweise durchgesetzt. Insbesondere bei der Gestaltung der Datenverarbeitungssysteme und hier speziell bei der Softwareentwicklung werden vorerst

Modelle von betriebswirtschaftlichen Sachverhalten erstellt, die den Focus zunächst nicht auf informationstechnische Details legen (z. B.: welche Klassen sind in der Datenverarbeitungssoftware erforderlich?), sondern in erster Linie fachliche Aspekte berücksichtigen (z. B.: welche Akteure spielen für welchen Prozess welche Rolle?). Der große Vorteil solcher, von informationstechnischen Details stark abstrahierender Modelle ist die Möglichkeit, technisch weniger versierte Fachvertreter in den Entwicklungsprozess mit einbeziehen zu können. Solche Modelle im Frühstadium einer Informationssystemgestaltung werden auch als fachkonzeptionelle Informationsmodelle bezeichnet [DELFMANN2006]. An dieser Stelle wird ein Problem der **Open-Model-Initiative** offensichtlich. Es wird nicht leicht zu bewerkstelligen sein, weniger versierte Fachvertreter dazu zu bewegen, aktiv an einem Entwicklungsprozess innerhalb der **Open-Model-Community** mitzuwirken und dementsprechend Feedback zu geben. Dadurch könnten wichtige Inputs beim Modellentwicklungsprozess unberücksichtigt bleiben.

Die nächsten Schritte entsprechen einer immer detaillierteren Konkretisierung des Informationssystems, indem immer mehr technisch spezifische Details in die Modellgestaltung einfließen. Datenverarbeitungs-konzeptionelle Informationsmodelle[5] werden in Hinblick auf die verwendete Implementierungsplattform erstellt und daraufhin in Implementierungs-Informationsmodelle[6] weiterentwickelt [vgl. SCHEER1992]. Das Ziel ist die fertige Informationstechnik (Abbildung 7).


**Abbildung 7: Informationssystementwicklung [vgl. SCHEER1992]**

[5] Konzeptionelles Modell (zitiert nach: [LAHRAY2006]): In einem konzeptionellen Modell beschreibt eine Klasse die konzeptionellen Gemeinsamkeiten von bestimmten Objekten, deren Rollen, deren Verwendung und deren Verantwortlichkeiten. Die erstellten Konzepte bleiben unabhängig von der Technologie, in der die Software realisiert werden soll. Sie beschreiben die Fachdomäne und nicht die Software.

Klassen dienen im konzeptionellen Modell dazu, die Begriffe, die Beziehungen und Prozesse der Fachdomäne zu kategorisieren und zu strukturieren . Die Aufgabe, das konzeptionelle Modell zu erstellen, wird als Analysephase bezeichnet. Sie wird häufig nicht von Softwareentwicklern, sondern von separaten Teams in Abstimmung mit den fachlichen Anforderern durchgeführt.

[6] Das Implementierungsmodell (zitiert nach: [LAHRAY2006]):Ein Implementierungsmodell legt die konkrete Umsetzung des konzeptionellen Modells fest. Manche Klassen aus dem konzeptionellen Modell können Klassen im Implementierungsmodell direkt entsprechen. Häufig wird es keinen direkten Bezug zu Klassen der Implementierung geben. Abhängig von gewählter Architektur und Programmiersprache werden bestimmte Konzepte unterschiedlich komplex realisiert.

Der **Metamodell-Browser** spielt in dieser Entwicklungskette in erster Linie bei der Erstellung der fachkonzeptionellen Informationsmodelle eine Rolle. Zu Beginn einer Modell-Entwicklung können sich die Entwickler der **Open-Model-Community** einen Überblick über bereits existierende (Referenz)-Metamodelle verschaffen. Diese könnten zum Beispiel die (Referenz-)Metamodelle aus der ADOxx-BPMS-Anwendungsbibliothek sein, die über ein Web-Service in den **Metamodell-Browser** geladen werden. Darauf aufbauend können die Entwickler nun entscheiden, inwieweit für die Entwicklung einer Lösung schon eine adäquate Modellierungssprache vorhanden ist, oder ob ein (Referenz)-Metamodell erst durch "Customizing" für die Entwicklung eines domänenspezifischen Informationsmodells adaptiert werden muss.

## 3.5 Referenzmodelle

Obwohl diese Arbeit vorwiegend Metamodelle thematisch behandelt, wird der Referenz-Begriff im Zusammenhang mit Modellen allgemein erörtert, da er für sämtliche Modellebenen synonym zu verstehen ist.

Allgemein wird unter Referenz eine "Empfehlung" oder ein "Verweis" verstanden. Ein Referenzmodell stellt somit ein Modell mit Empfehlungscharakter bzw. ein Modell auf das verwiesen werden kann, dar. In der Modellierung steht im Besonderen der Wiederverwendungsaspekt der Referenzmodelle im Vordergrund. Ein Referenzmodell soll als Ausgangspunkt für die Erstellung weiterer Modelle dienen bzw. soll es für eine domänenspezifische Adaptierung ein Grundgerüst anbieten.

DELFMANN kehrt vier der charakterisierenden Merkmale für Referenzmodelle, die in der Literatur diskutiert werden, besonders hervor [vgl. DELFMANN2006]:

- Allgemeingültigkeit;
- Branchenbezug;
- Vollständigkeit;
- Adaptierbarkeit;

Am Beispiel der **Metamodellierungsplattform von ADOxx** werden diese vier Merkmale analysiert:

**Allgemeingültigkeit** bedeutet, dass ein Referenzmodell als Basis für alle daraus abgeleiteten spezifischen Modelle herhalten muss. Dabei muss man hinterfragen, wie Allgemeingültigkeit

zu verstehen ist. Hier stehen sich zwei Betrachtungsweisen gegenüber. Auf der einen Seite kann man ein Referenzmodell als allgemeingültig bezeichnen, wenn der Detailgrad sehr niedrig ist und ein abgeleitetes Modell durch spezifische Details ergänzt werden kann. Andererseits kann man ein Referenzmodell, dessen Detailgrad schon sämtliche spezifischen Eventualitäten abdeckt, als allgemeingültig auffassen. Bei dieser Art des Referenzmodells werden die spezifischen Modelle durch Auswahl der relevanten, bereits vorhandenen Elemente erstellt werden.

Bei der **ADOxx-Metamodellierungsplattform** ist der Detailgrad der zur Verfügung stehenden Modelltypen als eher gering zu bezeichnen. Die Anpassung an ein spezifisches Metamodell wird durch das "Customizing"[7] ermöglicht, womit die Modellierungsmethode von ADOxx umfassend erweiterbar ist. Durch das "Customizing"-Prinzip auf Metamodel-Ebene kommt man dem Merkmal der Allgemeingültigkeit somit sehr nahe.

In der Literatur wird von einigen Autoren [MARENT 1995] eine konkrete Zuordnung eines Referenzmodells zu einer bestimmten Branche (**Branchenbezug**) gefordert. Dieser Branchenbezug würde bedeuten, dass ein Referenzmodell immer einer konkreten betriebswirtschaftlichen Domäne zuzuordnen ist. Dies würde im Umkehrschluss bedeuten, dass es keine allgemein gültigen bzw. branchenneutralen Referenzmodelle geben könnte. Der Branchenbezug bei Referenzmodellen von Geschäftsprozessen erscheint durchaus sinnvoll. Inwieweit ein Referenzmodell für eine Organisationsstruktur branchenspezifisch sein muss, bleibt aber zu hinterfragen. Prinzipiell bleibt es jeden Modellierer selbst überlassen, ob er einen Branchenbezug bei der Modellierung eines Referenzmodells herstellt oder nicht.

Die **ADOxx-Metamodellierungsplattform** weist bei den Basis-Modelltypen keinen erkennbaren Branchenbezug auf. Der Branchenbezug ergibt sich erst durch die spezifische Erweiterung bzw. Anpassung (Customizing) eines Referenzmetamodells an eine konkrete Domäne.

Unter **Vollständigkeit** versteht man die Eigenschaft, dass ein Referenzmodell ohne weitere Anpassung für die Modellierung zumindest **eines** betriebswirtschaftlichen Sachverhalts verwendet werden kann. In Bezug auf die Metamodellierung würde dies bedeuten, dass ein Referenz-Metamodell zumindest **eine** betriebswirtschaftliche Domain ohne weitere Anpassung repräsentieren können muss.

---

[7] Durch die ADOxx eigene Scriptsprache ADOScript, ist es möglich auf Metaebene die Modelltypen um nahezu jede beliebige Methode zu erweitern.

Die **Basis-Modelltypen von ADOxx** sind in ihrem Umfang ausreichend genug, um auch ohne domänenspezifische Anpassungen betriebswirtschaftliche Sachverhalte zu repräsentieren.

Des Weiteren gilt die **Adaptierbarkeit** eines Referenzmodells als wichtiges Kriterium. Ein Referenzmodell gilt dann als adaptierbar, wenn es eine methodische Unterstützung für die Ableitung der spezifischen Modelle gibt. Dieses Kriterium scheint nicht wirklich schlüssig, da auch ohne methodische Unterstützung ein Modell als Referenz dienen kann.

Wie schon zuvor erwähnt, ist das Customizing einer der großen Pluspunkte der ADOxx-Metamodellierungsplattform. Das Customizing ermöglicht umfassende Adaptionen der Basis-Modelltypen, die hier als Meta-Referenzmodelle angesehen werden, an die verschiedensten Domänen.

Die auf der ADOxx-Plattform hinterlegten Metamodelle stellen eine mögliche Quelle für den **Metamodell-Browser** dar, wobei die Möglichkeit besteht, über Web-Services jede x-beliebige freigegebene Plattform als zusätzliche Quelle von bereits existierenden Metamodellen zu erschließen. Indem diese Metamodelle als Ausgangsbasis für Modellentwicklungen dienen, können diese als Referenz-Metamodelle bezeichnet werden, die der **Open-Model-Community** zur Verfügung gestellt werden.

## 3.6 Metamodellierungskonzepte

Das folgende Unterkapitel erläutert die grundlegenden Konzepte der Metamodellierung. Als Ausgangspunkt dient das Metamodellierungskonzept von Karagiannis und Kühn [KARKÜHN2002].

### 3.6.1 Modellierungsmethode

Eine Modellierungsmethode (Abbildung 8) setzt sich aus zwei großen Komponenten zusammen. Das ist erstens die Modellierungstechnik und zweitens der Modellierungsmechanismus mit den dazugehörenden Modellierungsalgorithmen. Oft ist es nicht möglich, sämtliche Aufgaben eines Modellierungsproblems ausschließlich durch die Bereitstellung einer Modellierungstechnik zu lösen. Zusätzliche Regeln (z. B.: Einschränkungen, Ausnahmen und Zusammenhänge) für die Modellerstellung können durch die spezifische Formulierung von Mechanismen und Algorithmen definiert werden.

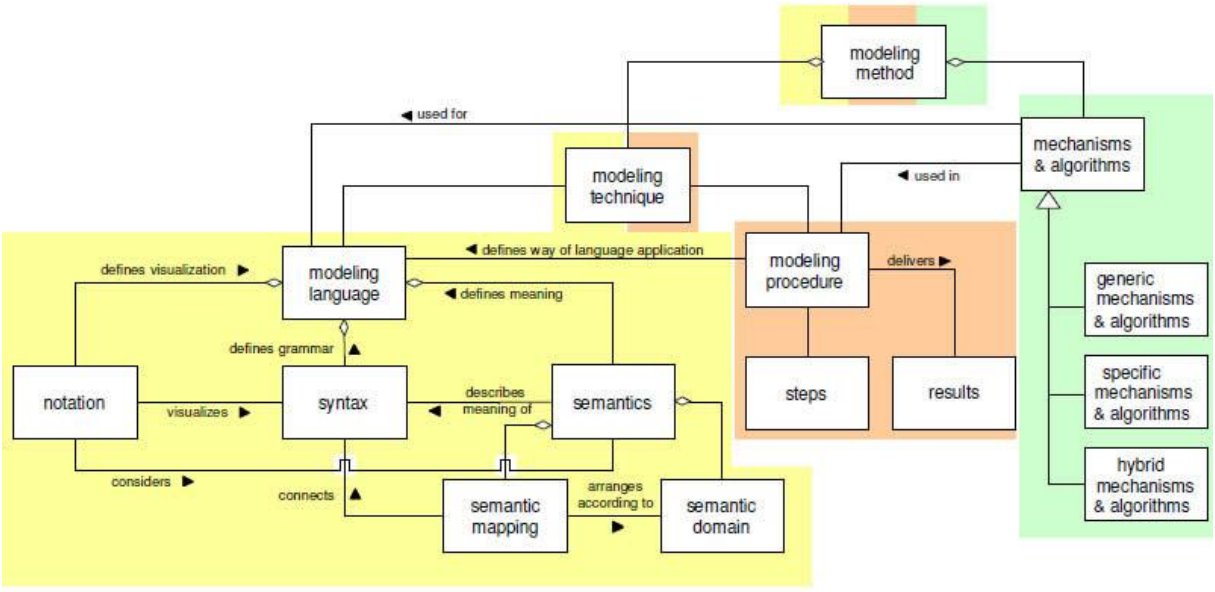


**Abbildung 8: Komponenten einer Modellierungsmethode [vgl. KARKÜHN2002]**

### 3.6.2 Modellierungstechnik

Die Modellierungstechnik setzt sich aus der Modellierungssprache und dem Vorgehensmodell (modelling procedure) zusammen. Im Vorgehensmodell werden die Schritte festgelegt, die bei der Erstellung einer Modellierungssprache die gewünschten Resultate erbringen sollen. Das Vorgehensmodell gibt sozusagen den Projektplan bzw. die Roadmap für die Definition einer Modellierungssprache vor. Exakt an diesem Punkt, bei der Definition der Modellierungssprache, ist der Einsatz des **Metamodell-Browsers** denkbar.

### 3.6.3 Modellierungsmechanismen und Modellierungsalgorithmen

Durch Mechanismen und Algorithmen wird die Funktionalität einer Modellierungssprache für den Gebrauch detaillierter definiert. Die Mechanismen können wie folgt unterschieden werden:

- **Generische Mechanismen** sind im Meta$^2$-Modell definiert. Sie können somit für alle Metamodelle verwendet werden, die von dem Meta$^2$-Modell abgeleitet werden.
- **Spezifische Mechanismen** werden konkret für ein bestimmtes Metamodell erstellt.
- **Hybride Mechanismen** sind zwar im Meta$^2$-Modell definiert, werden aber für ein bestimmtes Metamodell angepasst.

### 3.6.4 Modellierungssprache

Die Modellierungssprache dient als Grundlage für die Erstellung eines **Informationsmodells**. Man kann eine Modellierungssprache als ein System von Zeichen mit einem zugrunde liegenden Regelwerk, wie diese Zeichen zu verwenden sind, verstehen. Die verwendete Modellierungssprache gibt somit vor, welche Zeichen in einem Modell verwendet werden dürfen, und in welcher Beziehung diese Zeichen in einem Modell zueinander stehen dürfen.

Die Beschreibung der Modellierungssprache erfolgt in einem **Metamodell**, wobei das Metamodell wiederum durch eine eigene Modellierungssprache, der s. g. Metamodellierungssprache definiert ist. Die Meta-Beschreibungen lassen sich nach oben hin prinzipiell beliebig fortsetzen, führen aber zwangsläufig zu einem immer geringeren Detailgrad der Modellierung durch Erhöhung der Abstraktion. Bei welcher Ebene eine Modellierungshierarchie endet, bleibt dem Modellersteller (in Bezug auf diese Arbeit würde es sich um einen **Chef-Entwickler der Open-Model-Initiative** handeln) selbst überlassen, er sollte aber darauf achten, dass die erreichte Abstraktionsebene noch sinnvoll nutzbar ist. Eine gängige Modellierungshierarchie ist die Metamodell-Architektur mit vier Ebenen(Abbildung 9). Die unterste Ebene bzw. die Layer 0-Ebene stellt das Original (z. B.: ein **Informationssystem**) dar. Von diesem Original wird auf der Layer 1-Ebene ein Modell erstellt. Im Metamodell auf der Layer 2-Ebene werden die Modellierungskonzepte für die Modellinstanz auf der Layer 1-Ebene definiert. Auf der Layer 3-Ebene werden wiederum die Modellierungskonzepte für die Metamodellinstanz auf der Layer 2-Ebene konzipiert.

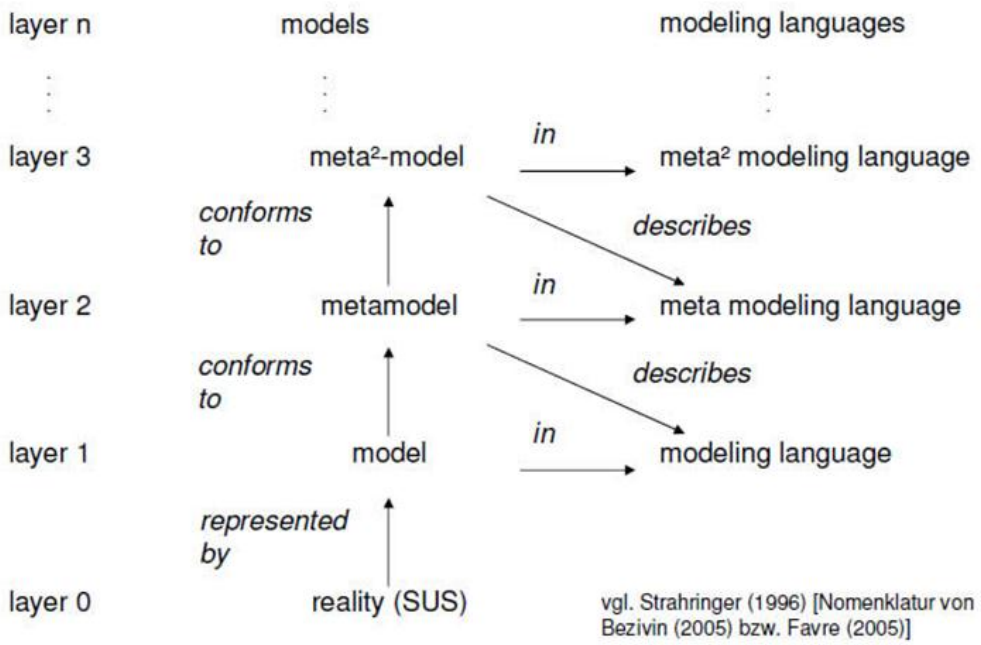


**Abbildung 9: Metamodellierungshierarchie [vgl. Vortrag Karagiannis 2007]**

Eine (graphische) Modellierungssprache wird durch ihre Syntax, ihre Semantik und ihre Notation beschrieben. Syntax und Semantik bilden den konzeptionellen Aspekt, in dem die Bedeutungen der Begriffe und deren Beziehungen innerhalb der Modellierungssprache bestimmt werden. Von den Begriffen ausgehend, können im Anschluss die zulässigen Modellierungselemente festgelegt werden.

Durch die **Syntax** werden die Elemente und die Regeln beschrieben, in welcher Form ein Modell erstellt werden darf. Die konkrete Definition der Syntax findet im Metamodell statt, wobei für die Darstellung eines Metamodells oft ein UML-Klassendiagramm verwendet wird. In Abbildung 10 ist auszugsweise das Metamodell eines Motorrads dargestellt. Die Elemente und deren Beziehungen untereinander ergeben die **Syntax**, wie ein Motorrad aufgebaut sein muss, um den Ansprüchen eines Motorrades zu entsprechen.

Für die Beschreibungen spezieller Einschränkungen finden so genannte "constraint languages" Verwendung, wie zum Beispiel die Object Constraint Language (OCL).

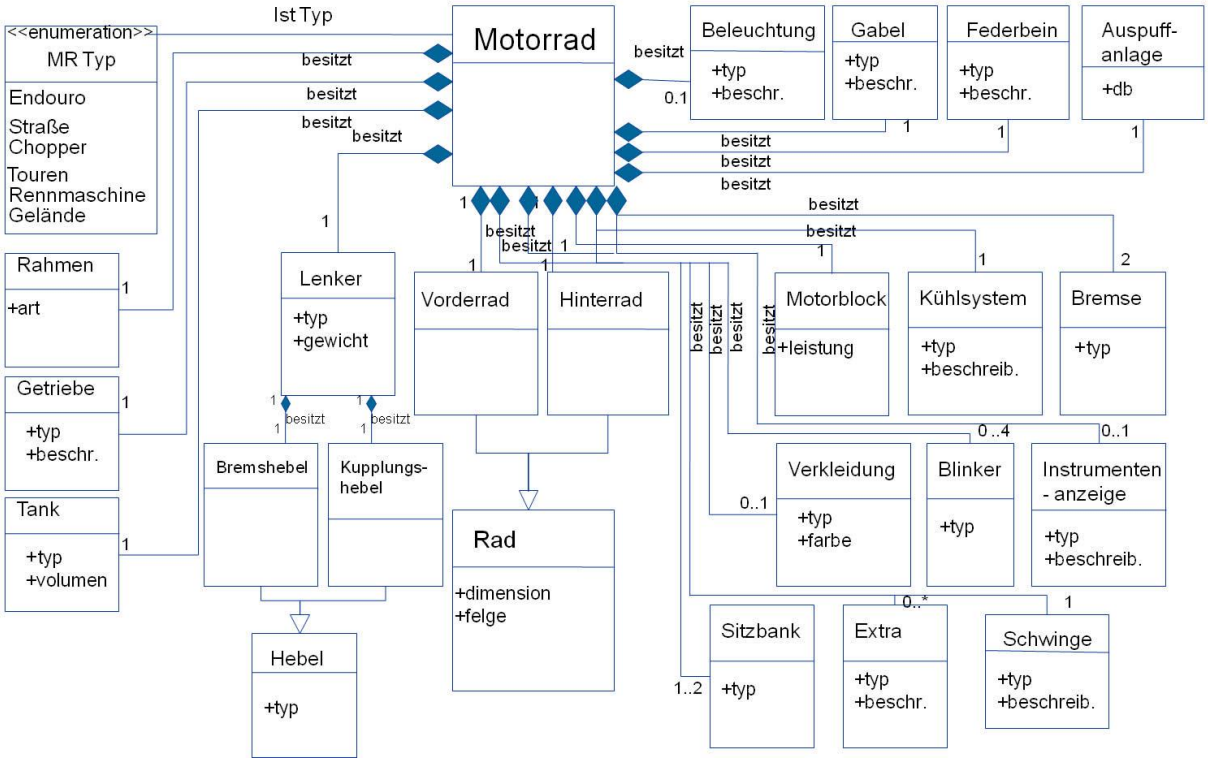


**Abbildung 10: Metamodell für ein Motorrad in Form eines UML-Klassendiagramms**

Durch die **Semantik** werden die möglichen Konstrukte einer Modellierungssprache definiert. Die Semantik fügt sich aus einer verwendeten semantischen Domäne und dem semantischen "Mapping" zusammen, wobei die Domäne eine spezifische Ontologie sein kann, und durch das semantische "Mapping" die Begriffe der verwendeten Ontologie mit den Elementen der Syntax durch ein semantisches Schema in Relation gesetzt werden. Somit wird eine Objektsprache definiert, deren Wahrheitsgehalt leicht zu überprüfen ist und ein breiter Konsens erzielt werden kann (siehe **semantische Theorie der Wahrheit** in Kapitel 2.2). Um semantische Definitionen zu formulieren, werden neben mathematischen Methoden auch reine Textbeschreibungen verwendet. Ein Beispiel dafür ist die semantische Definition von UML.

Bezugnehmend auf das zuvor gezeigte Metamodell eines Motorrades (Abbildung 10) könnte eine textuelle, semantische Definition auszugsweise wie folgt aussehen:

- Lenker ist vorne auf der Gabel montiert;
- Bremslicht ist am Heck montiert;
- Bremshebel auf rechter Lenkerseite montieren;
- Kupplungshebel auf linker Lenkerseite montieren;

- Gabel bildet Verbindung zwischen Lenker und Vorderrad;
- ...

Die **Notation** bildet den repräsentativen Aspekt und beschreibt die graphische Darstellung von Modellierungssprachen. Neben den zur Verfügung stehenden graphischen Symbolen und den zulässigen Konnektoren für die Verknüpfung der Symbole sind in der Notation auch Regeln für eine mögliche Layout-Gestaltung definiert. Die Notation hat aber prinzipiell keinerlei Einfluss auf die Funktion der Modellierungssprache. Wenn man sämtliche Symbole einer Modellierungssprache gegen andere Graphiken austauscht, wird dadurch keine neue Modellierungssprache begründet.

Erneut Bezug nehmend auf das Motorradbeispiel aus Abbildung 10 hier ein paar mögliche Notationen für definierte Elemente:

Lenker Rad Gabel

### 3.6.5 Modellierungsvorgehen

Die Modelling Procedure entspricht einem Vorgehensmodell und beschreibt die Vorgehensweise für die Verwendung einer Modellierungssprache, um mit der Modellierungssprache Modelle zu erstellen. Wie schon zuvor im Zusammenhang mit der Modellierungstechnik erwähnt, werden im Vorgehensmodell die Schritte festgelegt, um im Endeffekt die gewünschten Resultate bei der Modellerstellung zu erreichen.

# 4 Open Model Initiative

In dieser Arbeit wurde schon mehrfach auf die Open-Model-Initiative und ihrer Community als Adressaten hingewiesen. Deshalb soll im folgenden Kapitel die Open-Model-Initiative und deren Visionen und Ziele vorgestellt werden. Es gilt zu bedenken, dass sich die Open-Model-Initiative und ihre Community nach wie vor in einem frühen Stadium des Aufbaus befindet und demzufolge noch nicht weitreichend etabliert ist. Demzufolge sind die weiteren Erläuterungen als visionärer Soll-Zustand zu verstehen, der von einem konkreten Ist-Zustand noch ziemlich entfernt ist. Diese Arbeit soll somit auch einen Beitrag zur Etablierung der Open-Model-Initiative leisten. Die weiteren Erläuterungen stützen sich in erster Linie auf die formulierten Visionen und Ziele, die KARAGIANNIS in seiner „Open-Model-Feasibility-Study“ [KARIGIANNIS2008] als „Manifest“ formuliert hat.

Der Leitsatz der Open-Model-Initiative:

„Models for everyone.“

Lässt die Vision schon klar erkennen. Das Ziel ist somit nicht, Modelle ausschließlich einer kleinen Expertengruppe zur Verfügung zu stellen, sondern auch Interessenten, deren Wissen über die Modellierungsthematik nicht so weit gediehen ist, aber deren Interesse, etwas über die Thematik zu lernen, prinzipiell vorhanden ist. Das Ziel ist somit die Etablierung einer Community, die sich mit dem Entwerfen, der Pflege, der Modifizierung, der Verbreitung und der Analyse von Modellen beschäftigt.

Dabei soll es zu keinerlei Einschränkungen, den Begriff des „Modells“ betreffend, kommen, wofür auch das „Open“ in Open-Model-Initiative steht. Vielmehr soll ein breites inhaltliches und domänenübergreifendes Spektrum abgedeckt werden. Das bedeutet, dass die Open-Model-Initiative ihre Community-Mitglieder nicht nur in den Reihen der Modellierer für betriebswirtschaftliche Sachverhalte sucht, sondern auch für gänzlich andere Domänen offen ist, wie z. B.: Biologie, Chemie und Medizin.

In Anlehnung an den Open-Source-Gedanken verspricht sich die Open-Model-Initiative bei Modellen, die von mehreren Entwicklern in einem kooperativen Umfeld erstellt werden, einen deutlich höheren Qualitätsgrad, als wenn die Modelle nur von einem „Experten“ bearbeitet werden. Dieses Zusammenspiel aus Erfahrung und Fachkompetenz soll dazu führen, dass potentielle Fehler schon in einem frühen Entwicklungsstadium erkannt werden und fehlende

Aspekte bei der Modellerstellung nicht vernachlässigt werden. Werden die so erstellten (Referenz-)Modelle mit den dazugehörenden Metadaten in einem frei zugänglichen Repository zur Verfügung gestellt und können die Modelle auch leicht gefunden werden, kann dadurch ein hoher Grad an Wiederverwendung erzielt werden. Der Vorteil der Wiederverwendbarkeit liegt auf der Hand. Kosten, Ressourcen und Zeit können gespart werden, Fehlerquellen die bei jeder Neumodellierung auftreten, können minimiert werden. Durch die Freigabe der Metadaten ergibt sich eine hohe Flexibilität bei der Weiterentwicklung bestehender Modelle und deren Anpassung an spezifische Sachverhalte.

Die Open-Model-Initiative stellt sich selbst als ein Zusammenspiel mehrerer Bereiche dar (Abbildung 11).

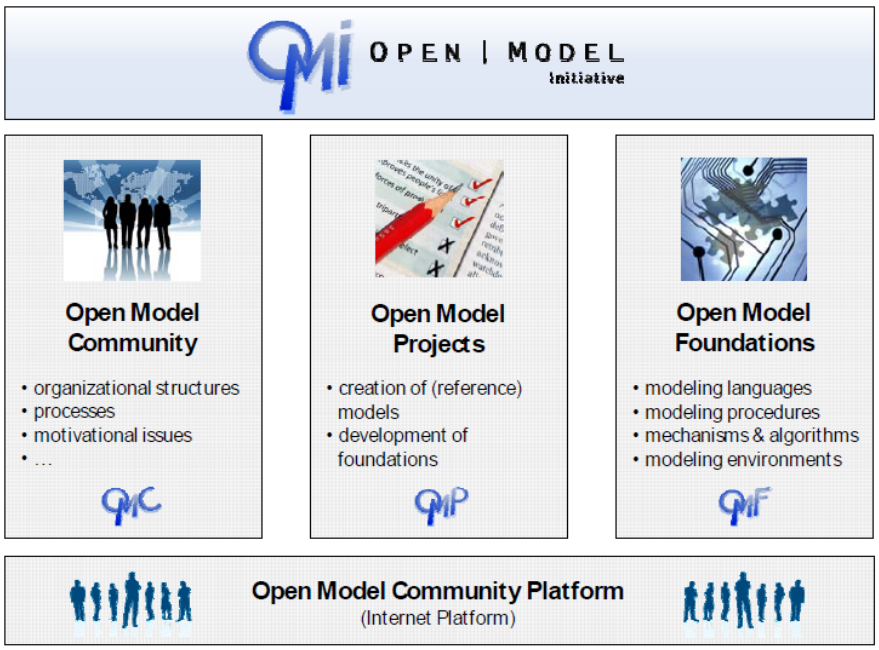


**Abbildung 11: Bereiche der Open Model Initiative [KARIGIANNIS2008]**

Aufsetzend auf die Open-Model-Community-Plattform, welche die Internet-Plattform repräsentiert und gleichzeitig als Basis betrachtet werden kann, teilt sie sich in drei Themengebiete.

## 4.1 Open Model Community

Im Allgemeinen werden Communities als Gruppen von Individuen bezeichnet, die gleiche Interessen teilen und zusammenwirken, um gemeinsame Ziele zu verfolgen. Für die Open-

Model-Community liegen die Hauptinteressen in der Verbreitung und der gemeinsamen Entwicklung von Modellen.

Obwohl die Open-Model-Community keine allzu strikten Definitionen für die Rollen innerhalb der Community festlegen will und sich eher am „Bazaar“-Gedanken orientieren will, wie ihn Raymond in seinem Buch „The Cathedral and the Bazaar“ [RAYMOND2001] geprägt hat, um die Struktur von Open-Source-Communities zu beschreiben, werden doch unterschiedliche Akteure und Rollen innerhalb der Community vorgeschlagen.

Als Teilhabende an der Open-Model-Initiative werden einerseits Unternehmen und andererseits Einzelpersonen unterschieden, deren Rollen durch Rechte und Pflichten definiert werden. In Anlehnung an die Softwareentwicklung werden zwei Arten von Rollen besonders hervorgehoben. Erstens die Anwender bzw. „User“, zu denen jeder zu zählen ist, der sich für ein Modell, ein Modellierungswerkzeug oder eine Modellierungstechnik interessiert und deshalb einen Blick auf die Community-Plattform riskiert, und zweitens die Entwickler bzw. „Developer“ die sich vorrangig mit der Entwicklung von Lösungen beschäftigen. Somit können zwar Entwickler auch immer Anwender sein, aber nicht jeder Anwender kann auch ein Entwickler sein. Zusätzlich kann man bei den Anwendern noch zwischen aktiven und passiven Anwendern unterscheiden und bei den Entwicklern zwischen Chef-Entwicklern und Co-Entwicklern differenzieren (Abbildung 12).

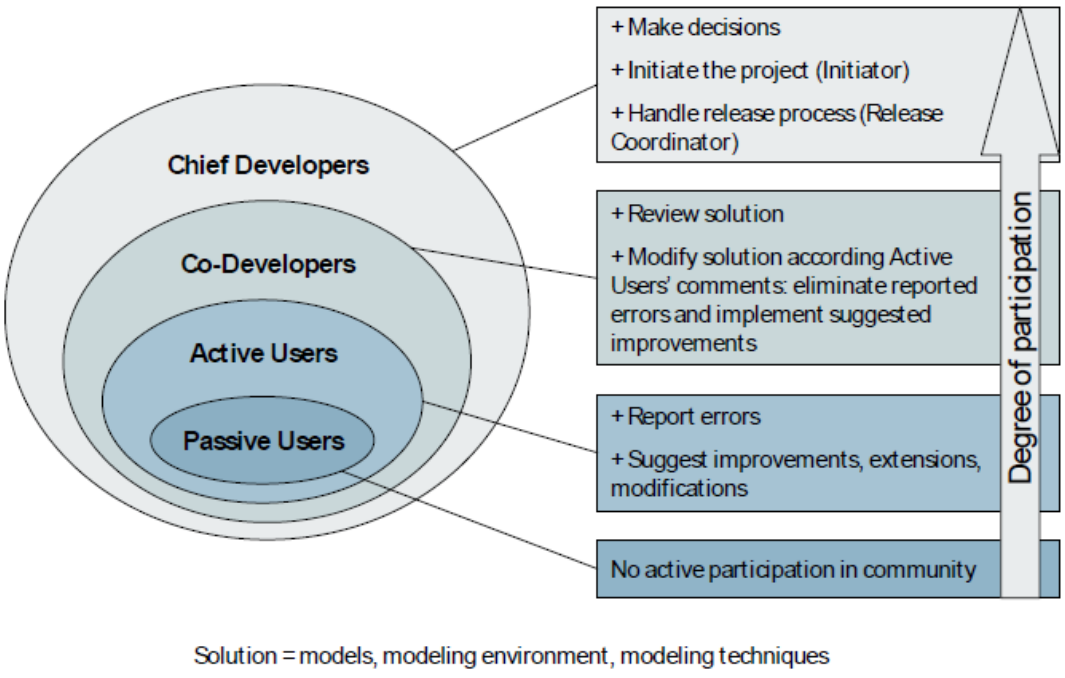


**Abbildung 12: Open Model Initiative – Rollen und Aufgaben [KARIGIANNIS2008]**

Im Unterschied zu den passiven Anwendern, die sich nicht aktiv am Entwicklungsprozess beteiligen, bringen sich aktive Anwender mit der Meldung von Fehlern und Verbesserungsvorschlägen bei Modellprojekten ein. Ein Co-Entwickler überprüft kritisch einen vorliegenden Lösungsansatz und modifiziert gegebenenfalls eine Lösung anhand von Vorschlägen, die durch aktive Anwender eingebracht werden. Der Chef-Entwickler kümmert sich darum, dass alle notwendigen Arbeiten im Projekt durchgeführt werden und trifft Entscheidungen, wie beispielsweise bestimmte Vorschläge und Fehlermeldungen erfasst werden sollen. Ein Chef-Entwickler kann auch als Initiator eines Projekts oder als Freigabe-Koordinator fungieren. Tritt er als Initiator auf, so ist er es, der die ursprüngliche Idee für den Entwurf einer neuen Lösung kreiert und auch der Mann, der das dazugehörende Projekt startet. Als Release-Koordinator wird ihm eine wichtige Rolle bei der Bestimmung des Entwicklungsstandes einer Lösung zu einem bestimmten Zeitpunkt zuteil und er entscheidet auch, wann eine Lösung offiziell freigegeben wird, ohne dass noch weitere Änderungen durchgeführt oder Fehler behoben werden müssen.

## 4.2 Rollen und Zuständigkeitsbereiche

Im Folgenden werden die wichtigsten Rollen und die ihnen zugedachten Zuständigkeitsbereiche überblicksmäßig zusammengefasst [vgl. KARAGIANNIS2008, S. 41-42]. Die Reihenfolge der Auflistung richtet sich dabei nach der chronologischen Abfolge, wie die einzelnen Akteure entsprechend ihrer Rollen in den zu erwartenden Projektablauf eingreifen werden, wobei die zuständige Rolle für einen Aufgabenbereich immer in Klammern nach der entsprechenden Aktivität gereiht ist. Anhand dieser Auflistung ist auch ersichtlich, welche Rechte welchen Rollen zugeteilt werden:

- Auslöser eines Projekts (Initiator): Ausschließlich Chef-Entwickler sind ermächtigt, Projekte in die Wege zu leiten bzw. zu starten. Von den Chef-Entwicklern geht die Idee für die Entwicklung einer Lösung aus, und sie sind es auch, welche die Umsetzung dieser Lösung anstreben.
- Meldung von Fehlern (Aktive Anwender, Co- und Chef-Entwickler): Sobald ein Entwickler oder aktiver Anwender einen Fehler, sei er syntaktisch oder semantisch, bzw. einen prinzipiellen Denkfehler in einem Modell entdeckt, so kann er bzw. soll er den Fehler auf der Projekt-Website melden.
- Hinweisen auf Verbesserungen (Aktive Anwender, Co- und Chef-Entwickler): Die Personen der betroffenen Rollen sind dazu aufgerufen, auf Verbesserungen

hinzuweisen bzw. zusätzliche Ideen einzubringen, wodurch ein Modell bei der Entwicklung einen höheren Nutzen erzielen kann. Solch eine Verbesserung könnte z.B.: eine mögliche Erweiterung des Modells sein oder die Einbeziehung eines neuen Elements der verwendeten Modellierungssprache.

- Überprüfung einer Lösung (Aktive Anwender, Co- und Chef-Entwickler): Durch die Überprüfung eines Modells durch die Personen der betroffenen Rollen kann überprüft werden, inwieweit ein Modell syntaktisch und semantisch korrekt ist. Sollte ein Missstand aufgedeckt werden, so wird dieser gemeldet und kann im Anschluss behoben werden.
- Modifizieren einer Lösung (Co- und Chef-Entwickler): Sobald von einem Fehler berichtet worden ist, muss das Modell von einem Entwickler entsprechend der gefundenen Fehler überarbeitet werden. Darüber hinaus können zusätzliche Verbesserungen und Erweiterungen in das Modell eingebracht werden.
- Entscheidungen treffen (Chef-Entwickler): Die Verantwortung über Inhalt, Design und andere wichtige Aspekte eines Modells trägt der Chef-Entwickler. Er entscheidet z. B.: ob eine gemeldete Verbesserung in der Entwicklung einer Lösung aufgenommen wird oder nicht.
- Koordination der Veröffentlichung (Freigabe-Koordinator): Sobald keine groben Bedenken mehr vorliegen, kann der Chef-Entwickler die Informationen zur Veröffentlichung eines Modells aktualisieren. So kann der Chef-Entwickler z. B.:, sobald keine Fehler mehr vorliegen und das Modell alle definierten Anforderungen erfüllt, den Status des Modells von „in Entwicklung“ auf „Freigabe“ setzen.

Zusammenfassend lassen sich als **Adressaten dieser Arbeit** in erster Linie die Chef- (und Co) Entwickler identifizieren, die Mitglieder der Open-Model-Community sind. Beim Starten eines Modellierungsprojekts sollen sie mit Hilfe des **Metamodell-Browsers** einschätzen können, ob es schon ein passendes **Referenzmetamodell** für die Lösung einer Problemstellung gibt bzw. welches Referenzmetamodell für die Lösung am besten geeignet ist. Ebenso wäre es auch denkbar, dass ein **vorhandenes Referenzmetamodell** als Ausgangsbasis für die **Adaption** einer vorhandenen Modellierungssprache Verwendung findet.

## 4.3 Open Model Foundations

Die Open-Model-Foundations repräsentieren die beiden wichtigsten Aspekte der Open-Model-Initiative in technischer Hinsicht. Das sind zum einem die Modellierungsmethoden und zum anderen die Modellierungsumgebungen. Für die Open-Model-Initiative sind diese Teilbereiche für eine erfolgreiche Etablierung des Open-Model-Gedankens existentiell wichtig. Erst wenn die angebotenen Modellierungsmethoden und Modellierungsumgebungen bzw. Modellierungswerkzeuge einen breiten Zuspruch in einer potentiellen Community finden, wird es einen nachhaltigen Erfolg für den Open-Model-Gedanken geben.

### 4.3.1 Modellierungsmethoden

Hinter einer Modellierungsmethode verbergen sich jene theoretischen Basistechniken, ohne deren Definition eine Modellerstellung nicht denkbar ist. Eine Modellierungsmethode besteht aus einer Modellierungssprache, einem Vorgehensmodell und den ergänzenden Mechanismen & Algorithmen, wobei die Modellierungssprache durch ihre Syntax, ihre Semantik und ihre Notation definiert ist. Da schon im Kapitel 3 „**Wissenschaftliche Grundlagen**“ auf diese Begriffe eingegangen wurde, sei an dieser Stelle nur auf das erwähnte Kapitel verwiesen.

Im Zusammenhang mit Modellierungsmethoden vertritt die Open-Model-Initiative den Standpunkt, dass das Hauptaugenmerk vorerst auf ein paar ausgesuchte Modelltypen beschränkt und die Modellpalette erst nach der erfolgreichen Implementierung einiger Projekte nach und nach erweitert wird. Dies geschieht ganz nach dem Motto: „Besser klein beginnen und guten Support für wenige Modelltypen bieten, als zu versuchen viele Modelle zu unterstützen, und die Qualität zu vernachlässigen“.

Bei der Auswahl der zu verwendenden Modelltypen zählt für die Open-Model-Initiative das schnelle, intuitive Verstehen eines Modells zu den wichtigsten Kriterien. Dadurch sollen möglichst viele Personen dazu motiviert werden, sich an der Initiative zu beteiligen. Ein weiterer Grund für die Wichtigkeit dieser Eigenschaften sind die angestrebten Adressaten. Wie schon zuvor erwähnt, will die Open-Model-Initiative nicht nur Experten ansprechen, sondern auch Personen mit geringerem Vorwissen, welche die Initiative als Ausgangspunkt für weitere Erfahrungen mit der Modellierungsthematik nutzen wollen. Aus diesem Grund wird die **Darstellung von Modellen in Diagrammform** als die beste Alternative für ein **intuitives Verständnis** von der Open-Model-Initiative angesehen. Der Informationsinhalt von einem Modell in Diagrammform, z. B.: das Entity-Relationship-Diagramm oder das Business-Process-Modell, sollte auch von einem unerfahrenen Modell-Anwender korrekt interpretiert

werden können, ohne dass er auf umfassende, zusätzliche Erklärungen angewiesen ist. Anhand dieser leicht verständlichen Modelle sollte somit auch ein breiter **Konsens** bei den Anwendern und ein **einheitliches Wahrheitsverständnis** erzielt werden. Demzufolge wird ein Modell von der Community erst dann als „richtig“ bezeichnet werden, wenn (fast) alle Subjekte innerhalb der Community ein Modell als „richtig“ bezeichnen. Die letzten beiden Sätze sind insofern von übergeordneter Wichtigkeit, denn sollte unter den Mitgliedern der Open-Model-Community, dem **konstruktivistischen Ansatz** folgend, **kein einheitliches Wahrheitsverständnis** erzielt werden, so wäre an ein **kooperatives Arbeiten** für eine bestimmte Modell-Lösung **nicht** zu denken.

Die Open-Model-Initiative geht bei der Wahl der vorerst unterstützten Modelle noch einen Schritt weiter und schränkt die zu verwendenden Modelltypen zusätzlich ein. Der Fokus wird auf drei Diagramm-Modelltypen festgelegt:

- **Datenmodelle** (z. B.: Entity-Relationship-Modelle, UML-Klassendiagramme, …) werden dazu benutzt, den Aufbau von Datenbanksystemen zu modellieren.
- **Prozessmodelle** (z. B.: Ereignisgesteuerte Prozessketten, UML-Aktivitätsdiagramme,…) dienen zur Abbildung der zeitlich-sachlogischen Abfolge von mehreren Einzelprozessen.
- **Wissensmodelle** (z. B.: Organigramme, Skillmodelle): beschreiben wichtige Aspekte von Organisationen. Durch Organigramme etwa kann graphisch der Aufbau einer Organisation dargestellt werden, Skillmodelle ordnen bestimmten Rollen erforderliche Fachkenntnisse zu.

Diese drei Modelltypen wurden gewählt, weil sie ein weites Feld von Domänen zu einem großen Teil modelltechnisch abdecken. Als Beispiele für Domänen werden von der Open Model Initiative genannt: Handel, der Finanzsektor, Logistik, Medizin, Industrieunternehmen, Universitäten usw. Das gleiche gilt für organisatorische Funktionen wie Einkauf, Verkauf, Produktion, Marketing, Finanz, IT usw..

Für die Entwicklung des **Metamodell-Browsers** ist die Positionierung der Open-Model-Initiative bei der Wahl der unterstützten Modelltypen nicht unwesentlich. Speziell bei der Visualisierung der Metamodelle muss auf die verwendeten Diagrammdarstellungen Rücksicht genommen werden. Es würde wenig Sinn machen, **für die Visualisierung** eine **Darstellungsform** zu wählen, die für Experten eine Einarbeitungszeit bedingt und für normale Anwender mit wenig Vorwissen unter Umständen gar nicht zu interpretieren ist.

Deshalb muss eine **graphische Notation** gewählt werden, die einerseits eine einfache Assoziation zu den Komponenten eines Metamodells zulässt und andererseits den Benutzer visuell soweit anspricht, dass er zum Weiterarbeiten animiert wird.

### 4.3.2 Modellierungsumgebungen

Der zweite technische Aspekt der Open-Model-Foundation beschäftigt sich mit der praktischen Umsetzung von Modellierungsprojekten. Um Modelle kooperativ zu erstellen und im Anschluss zu veröffentlichen, werden **Modellierungswerkzeuge** benötigt, die eine Vielzahl an Kriterien erfüllen müssen. Die Verfügbarkeit solcher Modellierungswerkzeuge für die Community ist ein entscheidender Faktor für die erfolgreiche Etablierung der Open-Model-Initiative.

Anhand zweier essentiell wichtiger Kriterien, das individuelle Anpassen von und das kooperative Arbeiten an Modellen, wurden aus 179!!! Modellierungswerkzeugen 17 ausgewählt, wovon acht Werkzeuge einer genaueren Betrachtung unterzogen wurden, weil sie entweder frei am Markt verfügbar sind oder als Trial-Version getestet werden können.

Da es im Zusammenhang mit der hier vorliegenden Arbeit **an Relevanz fehlt**, wird auf die Ergebnisse der Analyse **nicht weiter eingegangen**. Der **Metamodell-Browser** wird als Rich-Internet-Applikation umgesetzt und bezieht die relevanten Daten über Services aus Metamodell-Repositorien. Demzufolge findet kein Modellierungsprozess statt und dadurch sind auch Modellierungswerkzeuge in diesem Kontext nicht relevant. Für die Ergebnisse der Modellierungswerkzeug-Analyse sei auf [KARAGIANNIS2008] verwiesen.

### 4.3.3 Open Model Community Plattform

Die Open-Model-Community-Plattform bietet die technischen Voraussetzungen, welche die verteilte und kooperative Entwicklung von Modellen erst ermöglicht. Über die Plattform können die Community-Mitglieder ihre Erfahrungen austauschen. Ebenso können Modelle veröffentlicht und diskutiert werden und die Entwickler können sich Feedback für ihre Modelle durch die Community zukommen lassen.

Bei den Anforderungen an die Funktionalität der Community-Plattform, die teilweise schon umgesetzt ist[8], wurde in drei verschiedene Kategorien unterschieden. Die erste Kategorie betrifft die obligatorischen Funktionen, die eine offene Internet-Plattform zur Verfügung

[8] siehe: www.openmodels.at

stellen muss. Diese Funktionen umfassen die Möglichkeit sich auf dieser Plattform zu registrieren und den Up- und Download von Modellen und Anhängen für die weitere Diskussion. In einem weiteren Schritt können Übersichtslisten für die unterstützten Modellierungsmethoden und die Entwicklungswerkzeuge hinzugefügt werden. Zusätzlich ist noch eine Suchfunktion vorgesehen, die es einem Interessenten erleichtern soll, ein für ihn relevantes Modell schneller zu finden. Für die Suchfunktion sind aber ausführliche und aussagekräftige Metadaten für jedes Modell Voraussetzung. Im dritten und letzten Schritt werden der Community-Plattform nicht-funktionale Features hinzugefügt, die es den registrierten Mitgliedern ermöglicht, die Plattform an ihre persönlichen Bedürfnisse anzupassen, so zum Beispiel eine anpassbare Benutzeroberfläche, eine Rechtschreibüberprüfung oder einen Spam-Schutz.

Bei den Usern bzw. Anwendern der Plattform kann man wieder zwischen aktiven und passiven Usern unterscheiden. Die passiven oder auch anonymen User haben Zugang zu allen Modellierungslösungen und können sich diese auch downloaden und weiter verwenden. Was anonyme User nicht dürfen, ist das Kommentieren der Modelle und das Uploaden modifizierter Modelle. Diese Privilegien haben ausschließlich registrierte Anwender. Somit kann sichergestellt werden, dass die Qualität der angebotenen Modelle und Kommentare erhalten bleibt, da es sich bei den registrierten Usern ausschließlich um aktive Anwender oder Modellentwickler handelt.

Die Community Plattform ist für diese Arbeit von großer Bedeutung, da der **Metamodell-Browser als Teil dieser Plattform** vorgesehen ist. Da der Browser ausschließlich Informationen über vorhandene Metamodelle anbietet und nicht für die Weiterentwicklung von Modellen ausgelegt ist, würde es nicht viel Sinn machen, den Zugang nur registrierten Benutzern zu gewähren und passiven Usern vorzuenthalten. Vielmehr sollen sich auch unerfahrene Besucher der Plattform mit dem Aufbau und der „Natur“ eines Metamodells auseinandersetzen können. Inwieweit der Zugang zum Metamodell-Browser über einen einfachen Link ermöglicht oder der Browser über einen Frame direkt in die Community-Plattform integriert wird, bleibt dem Geschmack der Open-Model-Initiative überlassen. Wie der Aufbau und die technische Umsetzung der Plattform vorgesehen und teilweise schon realisiert wurden, zeigt Abbildung 13.

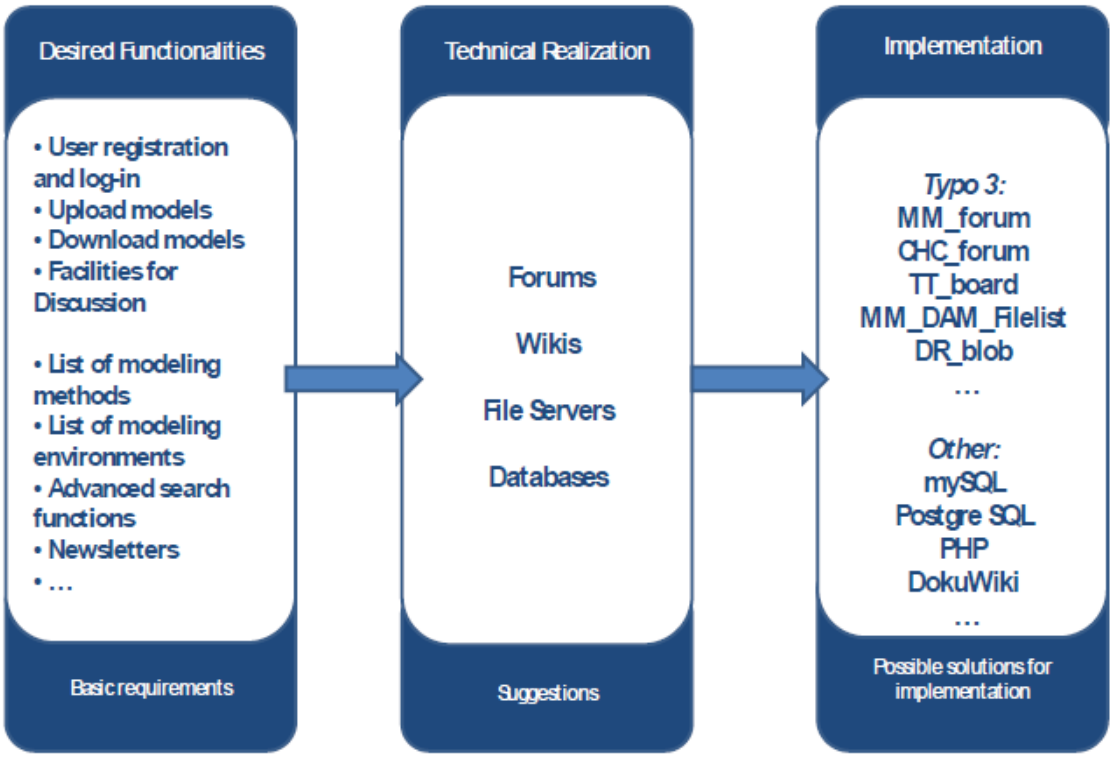


**Abbildung 13: Aufbau der Open Model Community Platform [KARAGIANNIS2008]**

### 4.3.4 Open Model Projects

Alle Aktivitäten, die auf der Community-Plattform durchgeführt werden, sollten innerhalb eines Projekt-Rahmens stattfinden. Ein Projektstart kann durch eine einzelne Person, eine Gruppe oder durch eine Organisation wie z. B.: Unternehmen und Universitäten, veranlasst werden. Die Open-Model-Initiative unterscheidet dabei drei verschiedene Typen von Projekten:

- **Modellierungsprojekt:** Die meisten Aktivitäten innerhalb der Initiative beschäftigen sich mit der Erstellung von neuen Modellinhalten. Diese Projekte bedienen sich bereits existierender Modellierungsmethoden und Modellierungsumgebungen, um Sachverhalte darzustellen, die für eine beliebige Domäne von Interesse sind. Ein wichtiger Aspekt ist auch die Entwicklung von **Referenzmodellen**, die für mannigfaltige Domänen und unterschiedliche Organisationen genützt werden können.
- **Projekte zur Methodenentwicklung:** Dieser Projekttyp beschäftigt sich mit der Erstellung neuer Modellierungsmethoden bzw. der Weiterentwicklung bereits existierender Methoden. In diesen Projekten werden Modellierungssprachen, Modellierungsprozeduren und Mechanismen & Algorithmen auf rein konzeptueller

Basis entwickelt. Die konkreten Implementierungen sind nicht Aufgabe dieser Projekte und werden anderenorts in eigenen Projekten durchgeführt.

- **Projekte für die Entwicklung von Modellierungsumgebungen:** In diesen Projekten werden Modellierungswerkzeuge erstellt und weiterentwickelt, die auf vorhandenen Modellierungsmethoden basieren. Der Rahmen für Projekte, die sich mit der Entwicklung von Modellierungsumgebungen beschäftigen, kann dabei sehr breit sein. Einerseits kann man komplett neue Software für die Modellerstellung entwickeln, wobei so ein Vorhaben mit einem beträchtlichen Aufwand verbunden wäre, andererseits kann man vorhandene Modellierungswerkzeuge weiterentwickeln und beispielsweise Erweiterungen erstellen, wodurch neue Modellierungssprachen und Prozessalgorithmen unterstützt werden.

Der Projektstart erfolgt immer durch einen Chef-Entwickler, der einen ersten Entwurf für eine Lösung, die innerhalb eines Projekts entwickelt werden soll, auf die Community-Plattform stellt und so den anderen potentiellen Projektteilnehmern der Open-Model-Community zugänglich macht. Sofern die Lösung für andere Mitglieder der Community von Interesse ist, werden sich die aktiven User in das Projekt einbringen, indem sie Fehler melden oder Verbesserungsvorschläge, Modifikationen und Erweiterungen anregen. Diese neuen Ansätze werden im Anschluss entweder direkt vom Chef-Entwickler aufgegriffen und umgesetzt, oder die Umsetzung geschieht durch die Aktiven User, die somit zu Co-Entwicklern werden. Für den Fall, dass sich mehrere Personen an dem Projekt beteiligen, empfiehlt die Open-Model-Initiative die Ernennung eines Release-Managers, der auf jeden Fall auch die Rolle eines Chef-Entwicklers innehat. Der Release-Manager entscheidet, wann ein Projekt die entsprechende Reife erlangt hat, um auf der Community-Plattform veröffentlicht zu werden, sei es als Release-Candidate-Version oder als fertige Release-Version.

Die Hauptgründe für die Trennung aller Aktivitäten in einzelne Projekte sind die bessere Überschaubarkeit und die leichtere Steuerung der Vorgänge innerhalb einer Lösungsentwicklung. Durch diese Abgeschlossenheit können Modelle entwickelt werden, ohne auf andere Projekte angewiesen zu sein. Es gilt aber zu bedenken, dass Abhängigkeiten zwischen Projekten immer wieder auftreten können. Es ist leicht vorstellbar, dass eine Person einen Sachverhalt modellieren will und plötzlich mit dem Problem konfrontiert ist, dass es noch keine geeignete Modellierungssprache für die Modellierung solch eines Sachverhalts gibt. In diesem Fall wird diese Person ein neues Projekt starten, um eine adäquate Modellierungssprache zu entwickeln. Ein weiteres Projekt könnte im Anschluss die neu

entwickelte Modellierungssprache in eine Modellierungsumgebung integrieren, z.B.: als Plug-In. Die Beziehungen der drei Projekttypen sind in Abbildung 14 ersichtlich.

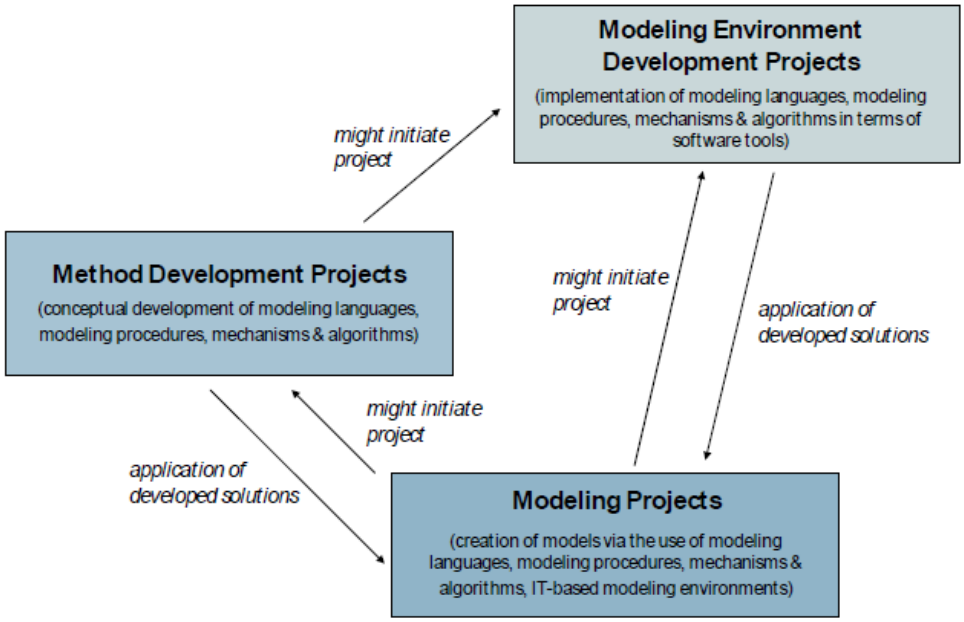


**Abbildung 14: Beziehung der Projekttypen [KARAGIANNIS2008]**

Speziell im Zusammenhang mit dem Starten eines Modellierungsprojekts soll der **Metamodell-Browser** eine wichtige Hilfestellung bieten. Der Browser soll die Person, die einen Sachverhalt modellieren will, bei der Überprüfung, ob bereits eine geeignete Modellierungssprache existiert, unterstützen. Wie bereits erwähnt, werden Modellierungssprachen durch Metamodelle beschrieben. Die Person hat im Anschluss die Möglichkeit, eine für den Sachverhalt passende Modellierungssprache zu wählen oder ein Projekt ins Leben zu rufen, das ein vorhandenes Metamodell durch „Customizing“ für den konkreten Sachverhalt anpasst. Sollte keine Modellierungssprache für den spezifischen Sachverhalt adaptiert werden können, muss ein Projekt für die Entwicklung einer neuen Modellierungssprache gestartet werden.

# 5 Rich Internet Applikationen

Da der Metamodell-Browser die dargestellten Metamodelle aus über Web-Services angebundene Metamodell-Repositorys bezieht, müssen die abgerufenen Inhalte am Web-Server programmtechnisch ausgewertet und im Anschluss für eine graphische Darstellung im Browser adaptiert werden. Des Weiteren soll die Darstellung im Browser den Anforderungen an das Erscheinungsbild einer Desktopanwendung entsprechen, aber keine Installation eines speziellen Web-Clients voraussetzen. Diese beiden Aspekte, Programmlogik auf dem Web-Server für die Datenaufbereitung und ansprechendes Erscheinungsbild, setzen die Verwendung einer Rich-Internet-Technologie voraus. Dieses Kapitel dient der Analyse und Bewertung von auf dem Markt verfügbaren Rich-Internet-Technologien. Anhand dieser Bewertung wird die Technologie gewählt, mit der die prototypische Implementierung des Metamodell-Browsers durchgeführt werden soll. Die zu betrachtenden Technologien sind AJAX/(X)HTML, das Google-Web-Toolkit, Flash/Flex von Adobe, Silverlight von Microsoft und JavaFX von SUN-Microsystems bzw. ORACLE.

## 5.1 Einführung

Zum ersten Mal wurde der Begriff Rich-Internet-Application von Jeremy Allaire in einem White Paper der Firma Macromedia 2002 verwendet [ALLAIRE2002]. Mittlerweile hat sich in diesem Bereich einiges weiterentwickelt. Macromedia wurde von Adobe übernommen, die ihrerseits den Rich-Internet-Gedanken weiterpflegen und mit der Flex-SDK, der AIR-Plattform und dem aktuellen Web-Office von Adobe-Acrobat.com immer wieder für innovative Produkte am Internetmarkt sorgen. In der Zwischenzeit versuchen auch Microsoft mit Silverlight und Sun mit JavaFX Technologien zu verbreiten, welche die Anforderungen an dieses rasch wachsende Marktsegment erfüllen. Zusätzlich gibt es noch einige Ajax-Frameworks, wie das Dojo-Toolkit, ExtJS und Jquery, die als Open-Source-Produkte zur Verfügung stehen.

Klassische Webdienste beruhen auf einem relativ simplen, aber bewährten Modell. Der Client bzw. der Webbrowser sendet eine Anfrage an einen Server, dort wird die Anfrage ausgewertet und eine neue aktualisierte Seite generiert. Im Anschluss schickt der Server die aktualisierte Seite zurück an den Client, wo diese im Browser neu geladen wird. Durch diese synchronen Aufrufe kommt es zu einem dauernden „Neu laden“ der Seite. Das ist einerseits langsam, andererseits ist der pausenlose Seitenwechsel sehr unkomfortabel für den Benutzer. Ein

weiteres Manko sind die eingeschränkten Funktionalitäten und Elemente, die vom aktuellen HTML-Standard 4.01 aus dem Jahr 1999 angeboten werden.

Mit der Verbreitung von JavaScript und AJAX und der damit verbundenen asynchronen Kommunikation, konnte die Bedienbarkeit von Webdiensten deutlich verbessert werden. Der klassische simple Webbrowser kann nun auch Teile der Programmlogik übernehmen und die graphische Benutzeroberfläche, d. h. die Webseite, muss nicht nach jeder Anfrage an den Server neu geladen werden. Der Client dient somit als Vermittler zwischen Browser und Server und ist für das Applikationsinterface und der damit verbundenen Kommunikation zum Server verantwortlich. Durch das Wegfallen des pausenlosen Seitenaufbaus werden Rich-Internet-Applikationen den handelsüblichen Desktop-Anwendungen im Aussehen und der Bedienbarkeit immer ähnlicher [vgl. SCHULZE2009].

Das Wort „Rich“ in Rich-Internet-Applikation soll auf den gesteigerten Mehrwert hindeuten, den diese Anwendungen unter anderem durch die Verwendung von „Rich Controls“ erlangen [MÜLLER2010]. Als „Rich Controls“ werden jene Bedienelemente bezeichnet, die in der aktuellen HTML-Spezifikation nicht vorhanden sind, z. B.: Rich-Text-Areas, Kalender, Datagrids, Slider, usw. (Abbildung 15).

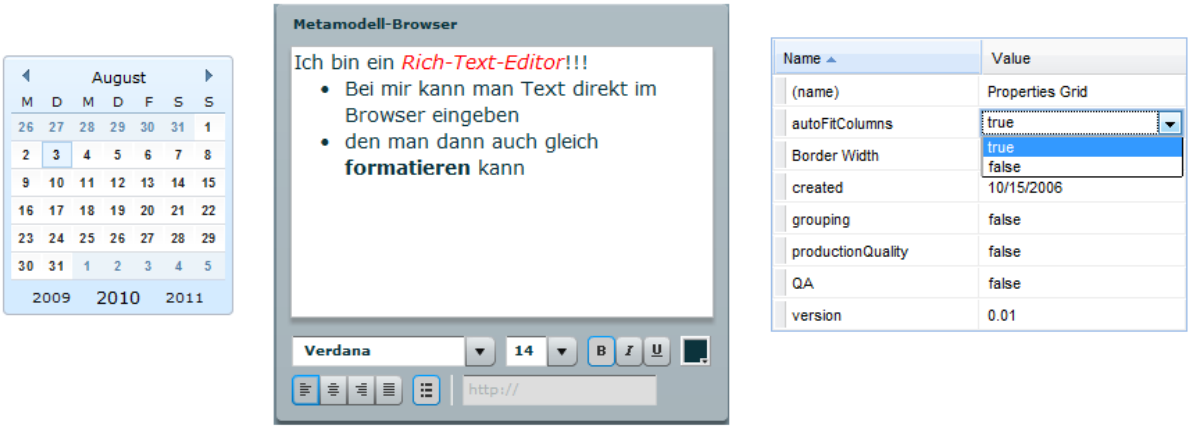


**Abbildung 15: s. g. Rich Controls**

Zusätzlich können HTML-Standardformularelemente in ihrem Funktionsumfang erheblich aufgewertet werden, so können z. B.: normale Formular-Texteingabefelder durch ein Autosuggestion-Feld[9], wie es unter anderem bei der Google-Suche Verwendung findet,

[9] Autosuggestion-Feld: Es werden dem Benutzer mögliche Suchbegriffe vorgeschlagen, indem Suchwörter automatisch vervollständigt werden.

ersetzt werden. Einfache Textareas können durch Rich-Text-Editoren eine Formatierung des Textes durch den Anwender, ähnlich wie in einem Textverarbeitungsprogramm, ermöglichen. Des Weiteren können Multimedia-Inhalte komfortabel in eine Webanwendung integriert werden und die Bedienbarkeit lässt sich durch Drag & Drop und Tastenkürzel erheblich intuitiver gestalten.

Ein weiterer Vorteil von Rich-Internet-Applikationen ist die Möglichkeit, Daten schon im Voraus zu laden, wenn davon ausgegangen werden kann, dass die Daten in einem der nächsten Arbeitsschritte benötigt werden. Der eben erwähnte Punkt im Zusammenhang mit der Durchführung von Berechnungen auf der Client-Seite erhöht die Performance und Leistungsfähigkeit eines Webdienstes enorm, da die Daten schon am Client zur Verfügung stehen und es dadurch nicht notwendig ist, eine zusätzliche Serveranfrage zu schicken.

Neben den genannten Vorteilen gibt es auch einige problematische Aspekte bei der Verwendung von Rich-Internet-Applikationen. Grundvoraussetzung ist ein moderner Browser und bei einigen Technologien auch die Installation von produktspezifischen Plug-Ins. Des Weiteren muss der Client-Rechner mit der zusätzlichen Ressourcenbelastung zurechtkommen und dementsprechend leistungsfähig sein. Rich-Internet-Applikationen setzen auch eine dauernde Verbindung mit dem Internet voraus, wobei dieser Aspekt speziell bei Auto- und Zugfahrten oder bei Flügen problematisch werden kann. Mittlerweile gibt es aber schon Technologien, die auch das Offline-Arbeiten zulassen, wie z. B.: Adobe AIR. Das Laden der Applikation dauert im Normalfall länger als bei traditionellen Webdiensten. Die asynchronen Anfragen, die Programmlogik und eine eventuell vorhandene Datenbank führen zu einer zusätzlichen Komplexitätsebene, welche die Entwicklung und das Debuggen der Applikation erschwert und zu zusätzlichem Aufwand beim Testen und für Performancemessungen führen kann.

Problematisch bei Rich-Internet-Applikationen ist die Verlinkung von Inhalten durch externe Links und auch die Verwendung des Zurück-Buttons bei Browsern führt oft nicht zum gewünschten Ergebnis. Das gleiche gilt für das Anlegen von Favoriten. Schwierigkeiten bereitet auch nach wie vor die Indizierung der Inhalte für Suchmaschinen. Dieses Problem hat schon viele Unternehmen davon abgehalten, ihren Web-Auftritt mit der Flash-Technologie

von Adobe zu realisieren. Mittlerweile wurde an diesen Problemen intensiv gearbeitet und es gibt schon brauchbare Lösungsansätze.[10]

## 5.2 Technologien

Rich-Internet-Technologien können grob in drei Kategorien einteilt werden. Das Kriterium bei dieser Kategorisierung liegt an dem benötigten Umfeld, welches die jeweilige Technologie voraussetzt [vgl. CAMPSTOCK2009, FLEX2011, DEWSBURY2008, JAVAFX2011, HANSONTACY2007, MÜLLER2010, SCHULZE2009, STEYER2009, WENZ2007, WIDJAJA2010]:

- **setzen keine Plug-Ins voraus**: Dabei handelt es sich vornehmlich um die klassische AJAX-Technologie. AJAX ist ein Zusammenspiel aus bewährten Websprachen, Standards und Techniken, die zu einer Rich-Internet-Technologie kombiniert wurden. Im Zusammenspiel mit ein paar JavaScript-Frameworks lassen sich durchaus ansehnliche Webdienste erstellen. Der große Vorteil von AJAX ist, dass alle verwendeten Komponenten (JavaScript, CSS, …) standartmäßig von allen aktuellen Browsern unterstützt und so keinerlei Zusatzinstallationen benötigt werden. Nachteil sind potentielle Browserinkompatibilitäten, die das Erscheinungsbild beeinträchtigen können. Abhilfe hat hier Google mit seinem Google-Web-Toolkit parat, auf welches noch zu einem späteren Zeitpunkt eingegangen wird (siehe Kapitel 5.2.2).
- **setzen Plug-Ins voraus**: Zu dieser Kategorie sind vor allem Flex von Adobe und Silverlight von Microsoft zu zählen. Will man Flex- bzw. Silverlight-Applikationen verwenden, so setzt dies ein browserspezifisches Plug-In voraus. Das bedeutet, dass bei der Arbeit mit unterschiedlichen Browsern für jeden Browser ein eigenes Plug-In installiert werden muss. Was sich im ersten Moment etwas umständlich anhört, hat einen riesigen Vorteil. Die Plug-Ins sorgen dafür, dass der Inhalt eines Web-Auftritts in jedem Browser zu 100% gleich dargestellt wird. Browserinkompatibilitäten, wie sie immer wieder auftreten und dazu führen, dass der gleiche Quellcode in verschiedenen Browsern zu einem gänzlich unterschiedlichen Erscheinungsbild führt, müssen somit nicht berücksichtigt werden.
- **können offline verwendet werden**: Diese Technologien ermöglichen sowohl die Interaktion mit einem Onlinedienst, können aber auch genutzt werden, wenn keine

[10] Z. B.: http:// www.flashseo.com

Internetanbindung zur Verfügung steht. Diese Rich-Internet-Lösungen, als Beispiele seien hier AIR von Adobe und Gears von Google genannt, setzen eine Installation des jeweiligen Dienstes auf dem Client-Rechner voraus. Sie werden deshalb als „Fat-Clients“ bezeichnet, im Gegensatz zu den beiden zuvor beschriebenen Kategorien, die man als „Thin-Clients“ bezeichnet und die, sieht man von den browserspezifischen Plug-Ins ab, keine weiteren Installationen auf dem Client-Rechner benötigen. Praktisch können mit diesen Technologien auch reine Desktop-Anwendungen realisiert werden, die einen gewaltigen Vorteil bieten: sie sind plattformunabhängig und können somit auf jedem unterstützten Betriebssystem verwendet werden. Ein weiterer Vorteil ist die Möglichkeit für Webentwickler, ohne weitere Einarbeitungszeit Desktop-Anwendungen erstellen zu können.

### 5.2.1 AJAX/(X)HTML

AJAX ist eine Kombination aus bewährten und etablierten Web-Technologien und steht für *Asynchronous JavaScript and XML* [FRIEDMAN2007]. Die AJAX-Architektur basiert auf folgenden Komponenten:

- **(X)HTML und CSS** dienen der Erstellung einer standartkonformen Seitenstruktur und Präsentation einer AJAX-basierten Anwendung.
- **DOM (Document Objekt Model)** ist für die dynamische Anzeige, Interaktion und Manipulation der durch (X)HTML und CSS erstellten Seitenstrukturen verantwortlich.
- Mit **XML (eXtensible Markup Language)** können externe oder lokal gespeicherte Daten geladen, ausgetauscht oder geändert werden.
- Die **XMLHttpRequest-Objekte** werden für den asynchronen Datenaustausch zwischen Client und Server und für die Generierung und Übermittlung von Anfragen benötigt.
- Mit der Skriptsprache **JavaScript** können die zuvor gelisteten Technologien miteinander verknüpft und in einer AJAX-Engine, die in jedem modernen Browser integriert ist, ausgeführt werden.

Durch die Kombination der Technologien können Websites dynamisch gestaltet werden. Das traditionelle Modell eines Webdienstes wird durch einen permanenten Vermittler zwischen Client und Server ergänzt. Beim ersten Laden einer Seite wird neben dem normalen (X)HTML-Dokument auch die bereits erwähnte AJAX-Engine mit geladen. Diese reagiert auf die Interaktionen eines Anwenders und schickt bei Bedarf Anfragen an den Server. Im

Anschluss werden die Antworten vom Server interpretiert und die Darstellung der Seite entsprechend angepasst. Die klassischen HTTP-Requests werden somit durch Engine-Anfragen ersetzt und die Inhalte der Seite aktualisiert, ohne dass die gesamte Seite komplett neu aufgebaut werden muss. Der große Vorteil dieser Technologie ist, dass die Engine in der Lage ist, mehrere Anfragen parallel zu bearbeiten und nicht eine Anfrage komplett abgearbeitet werden muss, bevor andere Anfragen durchgeführt werden können. Dieses Verhalten wird als Asynchronität bezeichnet und ist eines der Hauptmerkmale von AJAX. Durch die Integration der AJAX-Engine in allen aktuellen Browsern müssen keine weiteren Plug-Ins installiert werden. Der Unterschied zwischen einer klassischen und einer AJAX-Applikation ist in Abbildung 16 ersichtlich.

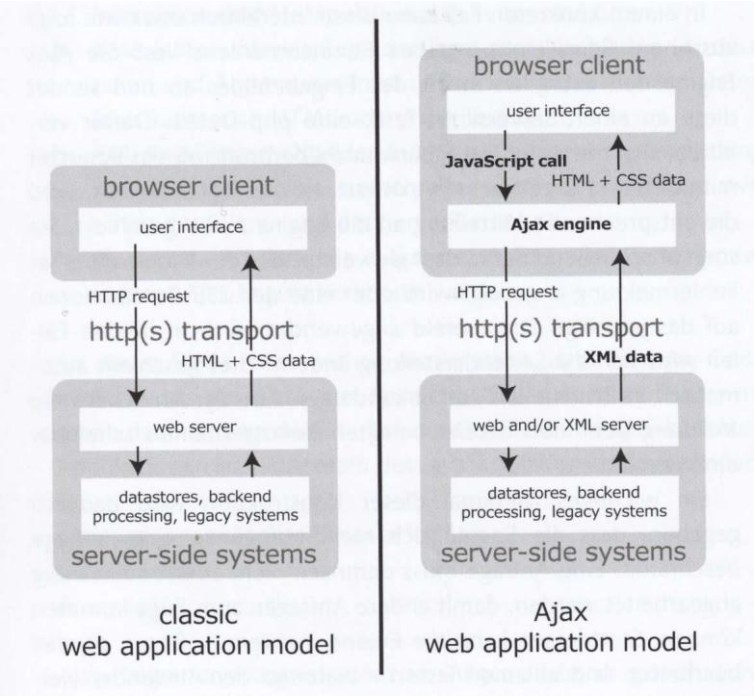


**Abbildung 16: Klassisches Modell einer Webapplikation vs. AJAX-basierten Ansatz [FRIEDMAN2007 S. 577]**

Die Funktionalität von AJAX kann durch die Einbeziehung von JavaScript-Bibliotheken wie z. B.: Jquery und ExtJS weiter gesteigert werden. **Jquery** bietet eine sehr spartanische Syntax und ähnlich wie bei objektorientierten Programmiersprachen können einzelne Methoden sehr einfach miteinander verkettet werden (Listing 1).

```
$(“div.test”).add(“p.quote”).addClass(“blue”).slideDown(“slow”);
```

**Listing 1: Beispiel Jquery-Syntax[11]**

In dem Code-Fragment werden allen div-Containern, die als class-Attribut „test“, und allen p (Paragraph)-Elementen, die als class-Attribute „quote“ haben, das class-Attribut „blue“ hinzugefügt und es wird eine slideDown-Animation in langsamem Tempo darauf ausgeführt.

Eine weitere JavaScript-Bibliothek ist **ExtJS** [EXTJS2011], die mittlerweile in der Version 3.2.2 frei verfügbar ist. Nebenbei erwähnt, dürfte es sich bei dem JS in ExtJS um die Initialen des Erfinders Jack Slocum handeln. ExtJS legt das Hauptaugenmerk auf User-Interface-Komponenten wie Toolbars, Trees, DataGrids, usw., welche die Bedienung eines Webdienstes intuitiver gestalten. Diese vorgefertigten Elemente werden auch als Widgets bezeichnet. Auf der ExtJS-Homepage ist als Beispiel ein Web-Desktop zu Anschauungszwecken abrufbar (Abbildung 17).

Große Erwartungen setzt die AJAX-Community in den, in Entwicklung befindlichen **HTML5** – Standard. Mit neuen Komponenten wie Canvas, Video, SVG etc. soll es möglich sein Animations- und Videoinhalte in einer Webapplikation zu integrieren, ohne dabei auf Plug-In-Lösungen wie Adobe-Flash oder Microsoft-Silverlight angewiesen zu sein. Dadurch würden unter anderem die Gestaltungsmöglichkeiten und somit auch die User-Experience von AJAX-Anwendungen wesentlich verbessert werden.[12]

[11] Vgl. http://de.wikipedia.org/wiki/JQuery, Stand: 04.08.2010

[12] Vgl. http:// www.chip.de/artikel/HTML5-Das-Web-von-morgen_41539437.html, Stand: 10.04.2011

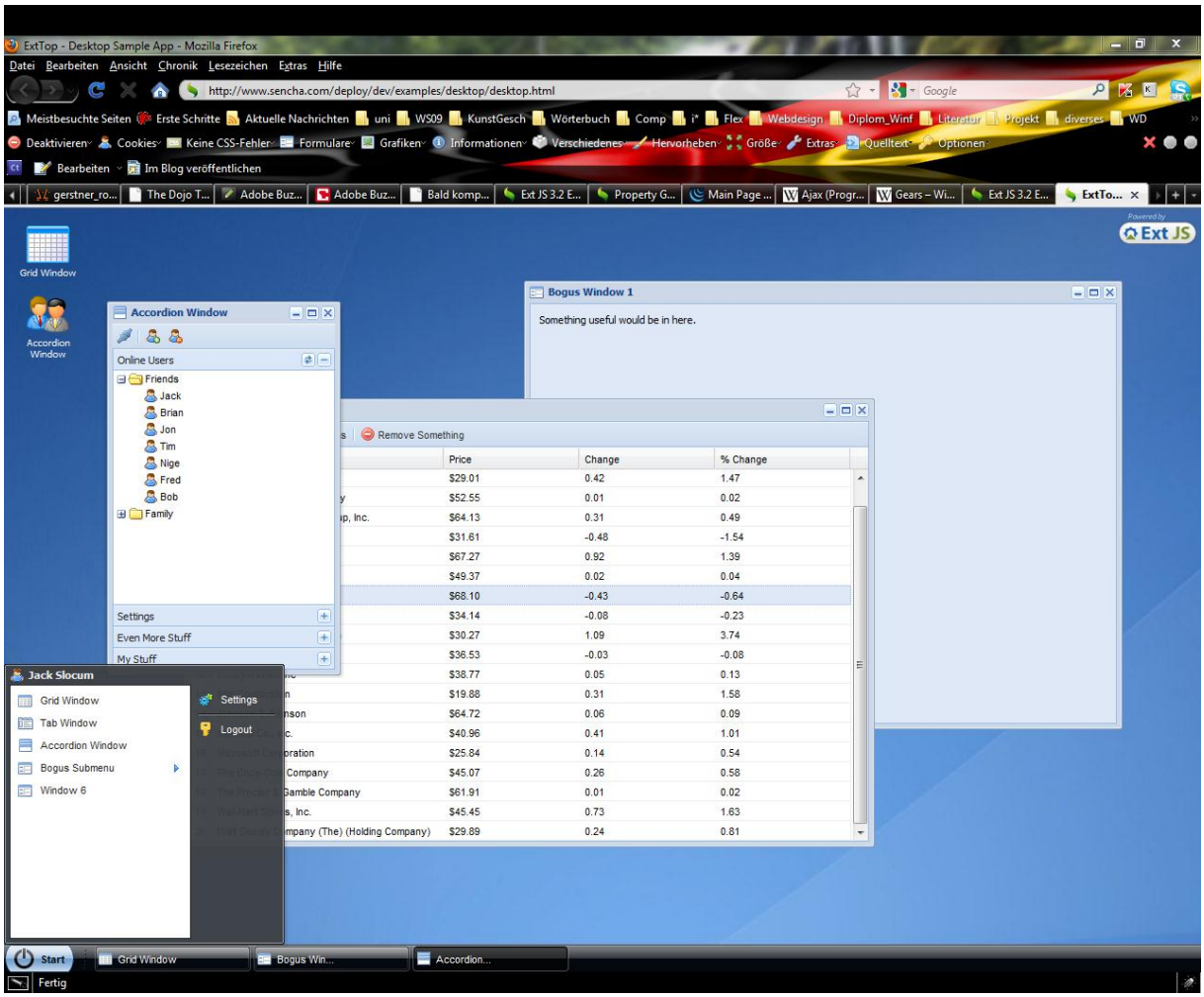


**Abbildung 17: Web-Desktop**

### 5.2.2 GWT – Google Web Toolkit

Google verfolgt seit 2006 mit seinem Google-Web-Toolkit einen völlig eigenen Weg bei der Erstellung von Rich-Internet-Applikationen. Die Internetanwendungen werden nicht in JavaScript programmiert, sondern in der objektorientierten Programmiersprache Java und im Anschluss mit Hilfe eines Java-JavaScript-Compilers in JavaScript, HTML und den dazugehörenden Cascading-Style-Sheets transformiert. Für die Ausführung der entwickelten Applikation ist somit keine Java-Virtual-Machine vonnöten. Dabei wird auch auf das Problem der Browserinkompatibilität Rücksicht genommen. Der Compiler generiert nicht nur eine JavaScript-Version der programmierten Java-Applikation, sondern er generiert für jeden gängigen Browser wie Internet Explorer, Firefox, Safari und Opera, eine eigene JavaScript-Version der Anwendung, wodurch eine einheitliche Darstellung in allen Browsern garantiert wird. Für den Internet Explorer geht der Compiler noch einen Schritt weiter und

berücksichtigt die verschiedenen Versionen des Browsers. Speziell Java-Entwickler können so ihrer bis dato favorisierten Programmiersprache treu bleiben und ersparen sich zeitintensive Um-Lernprozesse. Für Eclipse stellt Google ein eigenes Plug-In zur Verfügung.

Das Google-Web-Toolkit bietet eine Sammlung von Entwicklungswerkzeugen, Programmierhilfen und für die Gestaltung einer Benutzeroberfläche einen Satz der gängigsten Widgets, wie z. B.: Dropdownmenüs, Textfelder, Menüleisten, Baumstrukturelemente, usw.. Die Widgets können auf ähnliche Weise wie die Java-eigenen Swing-Elemente verwendet werden (Abbildung 18).

Bei der Arbeit mit dem Google-Web-Toolkit muss man jedoch nicht die gesamte Entwicklung auf Java-Code umstellen, es ist auch möglich, bereits vorhandenen JavaScript-Code in eine GWT-Applikation einzubinden. Für die Kommunikation zu einem Server kann auf das JavaScript-Objekt XMLHttpRequest über Wrapper zugegriffen werden. Zusätzlich werden Klassen zur Verfügung gestellt, die das JSON-Nachrichtenformat unterstützen [vgl. HANSONTACY2007 und DEWSBURY2008].


**Abbildung 18: Beispiel für GWT-Applikation**

Es werden zwei Modi bei der Applikationsentwicklung mit dem Google-Web-Toolkit unterschieden, zum einen der Hosted-Mode und zum anderen der Web-Mode. Im Hosted-Mode werden die Java-Applikationen als Bytecode von der integrierten Java-Virtual-Machine ausgeführt, wodurch das Debuggen einer GWT-Applikation wie bei jedem anderen Java-Programm möglich ist und man sich die Vorteile und Funktionen des Java-Debuggings

bei der Programmentwicklung zu Nutze machen kann. Zusätzlich ist ein Applikationsserver in Form eines Apache-Tomcat  im Toolkit für die Ausführung der Applikationen enthalten.

Im Web-Mode werden die Applikationen mit dem integrierten GWT-Compiler kompiliert und die HTML, CSS und JavaScript-Dateien spezifisch für den jeweiligen Browser generiert. Nur die generierten Dateien werden im Anschluss den Anwendern auf einem Webserver zu Verfügung gestellt.

Die GWT-Architektur umfasst folgende Komponenten (Abbildung 19):

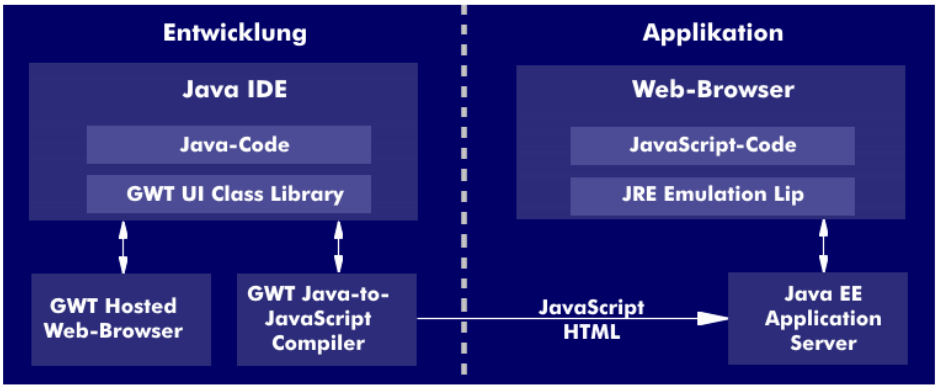


**Abbildung 19: GWT-Architektur [ITWISSENGWT2011]**

- Die wichtigste Komponente ist der **Java-to-JavaScript-Compiler**. Dieser ist für die Übersetzung des Java-Programmcodes zuständig und erzeugt die Dateien, die schlussendlich im Web-Mode ausgeführt werden können. Die generierte statische HTML-Seite dient dabei nur als Start-Container für die Applikation.
- Der **Hosted-Web-Browser** dient zur Entwicklung der GWT-Applikationen und ermöglicht die Darstellung der Inhalte im Hosted-Mode, ohne zuerst das Java-Programm in JavaScript kompilieren zu müssen.
- Die **UI-Class-Library** beinhaltet die Schnittstellen und Klassen, mit denen man Widgets erstellen kann, wie bereits zuvor erwähnt.
- In der **JRE-Emulation-Library** befinden sich JavaScript-Implementierungen der java.lang- und java.util-Pakete aus der Java-Standard-Klassen-Bibliothek, wobei nur einige Klassen dieser beiden Pakete unterstützt werden. Andere Java-Pakete aus dem Java-Developement-Kit werden bis dato nicht unterstützt.

Die Integration von JavaScript innerhalb des Java-Programmcodes kann durch das JavaScript-Native-Interface realisiert werden. Dadurch können JavaScript-Erweiterungen in einer

Applikation verwendet werden, deren Umsetzung unter Umständen nicht durch die Java-Bibliotheken möglich ist, wodurch sich der Vorteil, keine expliziten JavaScript-Kenntnisse bei der Entwicklung einer AJAX-Applikation besitzen zu müssen, wieder relativiert. [vgl. ITWISSENGWT2011]

### 5.2.3 Adobe-Flex

1997 wurde von der Firma Macromedia das Animationswerkzeug Flash auf den Markt gebracht. Ziemlich bald machte es den Anschein, als hätte Flash neben der Erstellung von reinen graphischen Animationen auch das Potential, als Grundlage für die Entwicklung webbasierter Applikationen zu dienen. Durch die immer weitere Verbreitung des Flash-Players und der Einführung der Skriptsprache ActionScript war eine Gestaltung von browserbasierten Anwendungen möglich, die mit rein HTML-basierten Anwendungen nicht denkbar waren. Wirklich durchsetzen konnte sich Flash als Internet-Technologie aber nicht. Ausschlaggebend waren mehrere Gründe. Zum einen hatte Flash in der Wirtschaftswelt lange Zeit das Image effektüberladener Darstellungen von Webinhalten, die eher mit nervenden Intros in Verbindung gebracht wurden als mit einer ernstzunehmenden Webtechnologie. Zum anderen war die Indexierung der Inhalte durch Suchmaschinen ebenfalls sehr lange ein ungelöstes Problem. Das größte Manko dürfte aber die Entwicklungsumgebung von Flash gewesen sein, die in ihrer Machart eindeutig mit einem Graphikprogramm in Verbindung gebracht wird und nicht mit einem Entwicklungswerkzeug für Web-Anwendungen. Das Animationsprogramm hat unumstritten große Ähnlichkeiten mit Graphik und Videoprogrammen wie Photoshop, Illustrator und After-Effects. Letzterer Umstand dürfte viele Webentwickler davon abgehalten haben, Flash als ernstzunehmende Alternativtechnologie zu erkennen. Um dieser Problematik entgegenzuwirken, wurde ein Framework inklusive einer IDE entwickelt, welche den Ansprüchen der Entwickler eher entsprachen, obwohl im Endeffekt die gleiche Technologie verwendet wurde. Das Resultat dieser Entwicklung war das Open-Source-Framework Flex. Zur Verfügbarkeit und Darstellung von Flexinhalten sei noch erwähnt, dass Flex zu jenen Web-Technologien gehört, die ein Browser-Plugin voraussetzen, den Flash-Player. Die Verbreitung des Flash-Players ist in Zeiten von „YouTube“ aber soweit fortgeschritten, dass man beinahe von einer Standartinstallation auf Desktop-Computern sprechen kann.

Die erste Flex-Version erschien im Jahr 2004. Nach der Übernahme von Macromedia durch Adobe wurde weiter an der Verbreitung der Technologie gearbeitet. Mittlerweile ist Flex in

der Version 4.0 unter einer Open-Source-Lizenz erhältlich. Das Flex 4 Software-Developement-Kit enthält das gesamte Klassen-Framework und den Compiler für die Generierung der ausführbaren SWF-Dateien, was bedeutet, dass eine Flex-Applikation in einem beliebigen Editor programmiert und im Anschluss über die Commandline kompiliert werden kann. Ebenfalls unter einer Open-Source-Lizenz (Mozilla Public License) ist die BlazeDS-Technologie frei erhältlich.

**BlazeDS** stellt alle Werkzeuge und Funktionen zur Verfügung, die für die Verbindung zwischen einer Flex-basierten Benutzeroberfläche und einer Java-basierten Programmlogik auf dem Server erforderlich sind. Der große Vorteil von BlazeDS wird bei der Datenübertragung erkennbar. Web-Browser sind nicht in der Lage, Java-Objekte direkt zu verarbeiten, deshalb müssen Java-Objekte in eine für den Browser verständliche Form umgewandelt werden. Entweder man transformiert die Daten eines Java-Objekts in einen Browser-verständlichen HTML-String oder man verwendet Techniken wie z. B.: JSON (JavaScript Object Notation) und wandelt das Java-Objekt in ein JSON-Objekt um. Es gibt auch noch weitere Möglichkeiten, Objekte vom Server zum Client zu transferieren, allen ist jedoch gemeinsam, dass ein Objekt von Java-ByteCode in ein für den Client verarbeitbares Format gebracht werden muss. Dieser Vorgang wird als **Serialisierung** bezeichnet und nimmt Ressourcen in Anspruch, die bei einer großen Menge von Anfragen beachtliche Ausmaße annehmen können. Ändert ein Anwender auf der Benutzeroberfläche Daten und schickt sie anschließend zurück an den Server, müssen die Daten am Server wieder in Java-Objekte transformiert werden, welche die Programmlogik verarbeiten kann. Dieser umgekehrte Vorgang wird als **Deserialisierung** bezeichnet und nimmt ebenfalls Ressourcen in Anspruch.

BlazeDS bedient sich bei der Serialisierung und Deserialisierung dem Action-Message-Format-3, kurz AMF3. Mit AMF3 wird ein Java-Objekt bei der Serialisierung mit nativem C-Code am Server in einen Binär-Stream umgewandelt. Die Deserialisierung wird anschließend im Flash-Player ebenfalls durch nativen C-Code durchgeführt und der Binär-Stream wird direkt in ActionScript-Objekte transformiert, die sofort in der Applikation verwendet werden können, um damit Tabellen oder Bäume mit Daten zu füllen. Java-Objekte können somit direkt in ActionScript-Objekte umgewandelt werden, womit der objektorientierte Ansatz auch beim Client konsequent weitergeführt werden kann [FLEX2011].

Als Entwicklungsumgebung bietet Adobe den kostenpflichtigen Flex-Builder an, der in der neuesten Version 4 offenbar als Reminiszenz an frühere Zeiten als Flash-Builder (Abbildung 20) bezeichnet wird. Der Flash-Builder basiert auf der Eclipse-IDE, die speziell im Java-

Umfeld weit verbreitet ist. Der Flash-Builder kann dadurch nicht nur als reines Oberflächengestaltungs-Tool verwendet werden, sondern kann auch auf die bekannten Funktionalitäten von Eclipse zurück greifen, d. h. es ist möglich, sowohl die Benutzeroberflächen mit der dahinter befindlichen ActionScript-Logik zu gestalten, als auch die Java-Business-Logik innerhalb der Flash-Builder-IDE zu entwickeln. Als zusätzliche Option bietet Adobe ein reines Eclipse-PlugIn des Flash-Builders an. Der Vorteil ist, dass sämtliche IDEs (BPEL, GWT, J2EE, PHP, Flex, usw.) die für die Entwicklung von Applikationen von Nöten sind, unter einem Dach vereint werden können, sofern man das will.

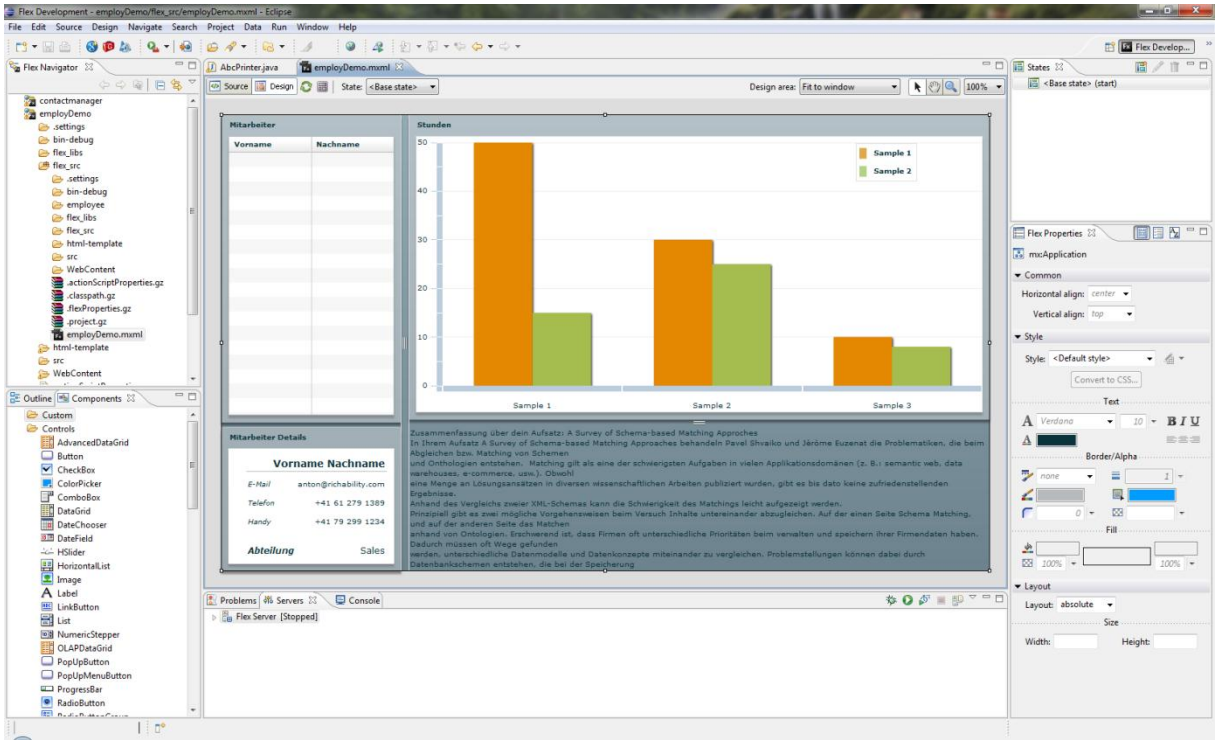


**Abbildung 20: Screenshot – Applikationsentwicklung in der Flash-Builder-IDE**

Das Rückgrat von Flex bilden MXML und ActionScript. MXML basiert auf XML und dient zum Erstellen des Layouts einer Benutzeroberfläche. In der Funktion ist MXML mit HTML, das bei traditionellen Web-Anwendungen Verwendung findet, vergleichbar. MXML liegt ein Satz von ActionScript-Klassen zugrunde, welche diverse UI-Komponenten zur Verfügung stellen. Der Grund für die Abstraktion der graphischen Darstellung durch MXML liegt in der leichteren Lesbarkeit für den Entwickler und der konsequenten Trennung von Layout und GUI-Programmlogik, Zweitere wird durch ActionScript realisiert. ActionScript nimmt die Rolle von JavaScript ein, das vor allem bei klassischen AJAX-Applikationen Verwendung findet. Des Weiteren ist ActionScript eine objektorientierte Skriptsprache, die es einem

Entwickler ermöglicht, umfassende Teile der Business-Logik auf dem Client zu implementieren. So können zum Beispiel Event-Handler, Listener, Objekte und Schnittstellen mit ActionScript realisiert werden. Sobald die Benutzeroberfläche mit MXML und die Programmlogik mit ActionScript entwickelt worden ist, kann der Compiler eine für den Browser interpretierbare SWF-Datei erzeugen, vorausgesetzt, der Browser hat das entsprechende Flash-Player-PlugIn installiert. [vgl. MÜLLER2010 und WIDJAJA2010]

### 5.2.4 Microsoft – Silverlight

Als Reaktion auf die zunehmende Verbreitung von Adobe Flex entwickelte Microsoft das RIA-Framework Silverlight, welches 2006 veröffentlicht wurde. Die Ähnlichkeit zu Adobe Flex ist nur schwer zu übersehen. Auch bei Silverlight ist die Installation eines PlugIns für den entsprechenden Web-Browser notwendig, und bei der Trennung von Oberflächengestaltung und Programmlogik ist ebenfalls eine starke Anlehnung an das Flex-Framework zu erkennen. Der offensichtliche Unterschied besteht lediglich darin, dass als XML-Derivat XAML (statt MXML) für die Gestaltung der Benutzeroberfläche verwendet und die Programmlogik durch eine der .NET-Sprachen, wie z. B.: C# oder VB (und nicht in ActionScript) realisiert wird. Browser-PlugIns gibt es mittlerweile für Internet-Explorer, Mozilla Firefox und Safari. Des Weiteren gibt es eine Open-Source-Erweiterung für Linux namens Moonlight und obwohl Opera nicht offiziell als Browser-Plattform unterstützt wird, ist es zwischenzeitlich bereits möglich, in diesem Browser Silverlight-Inhalte zu nutzen.

Als technologische Grundlage für Microsoft Silverlight dient das mit dem .NET-Framework-3.0 eingeführte Graphik-Framework WPF (Windows Presentation Foundation). WPF gilt als Nachfolger von Windows-Forms, und wird in erster Linie, ähnlich wie Adobe Flash, für die Darstellung von Vektorgrafiken verwendet, wobei auch Bitmap-Graphiken unterstützt werden. Um Benutzeroberflachen gestalten zu können, unterstützt WPF 2D- und 3D-Funktionalitäten und stellt eine Bibliothek mit vorgefertigten Steuerelementen zur Verfügung. Wurden Windows-Forms-Anwendungen noch direkt im Quellcode erstellt, so wird mittlerweile das schon zuvor erwähnte XML-Derivat XAML für die Gestaltung der graphischen Oberfläche verwendet. WPF-Anwendungen sind reine Desktop-Applikationen und sind nur in einem Windows-Umfeld lauffähig und dieses mit der zusätzlichen Einschränkung, dass die Windows-Version das .NET-Framework 3.0 unterstützt. Demzufolge gilt als Mindestanforderung die Installation von Windows XP Service-Pack 3 als Betriebssystem.

Silverlight ist der webfähige Ableger von WPF, mit dem nun auch die Erstellung von Rich-Internet-Applikationen ermöglicht wurde. Der Codename bei der Entwicklung von Silverlight war WPF/E (Windows Presentation Foundation / Everywhere) und weist somit schon auf die technologischen Wurzeln hin. In der relativ kurzen Geschichte von Silverlight ist bereits die vierte Version erschienen. Daran lässt sich erkennen, welches Potential Microsoft im Marktsegment für Rich-Internet-Applikationen erkannt hat. [CAMPSTOCK2009]

Das Silverlight-Framework [SILARCH2011] besteht aus dem Präsentationsframework, dem .NET Framework für Silverlight und zusätzlich aus Installations- und Updateprogrammen (Abbildung 21).

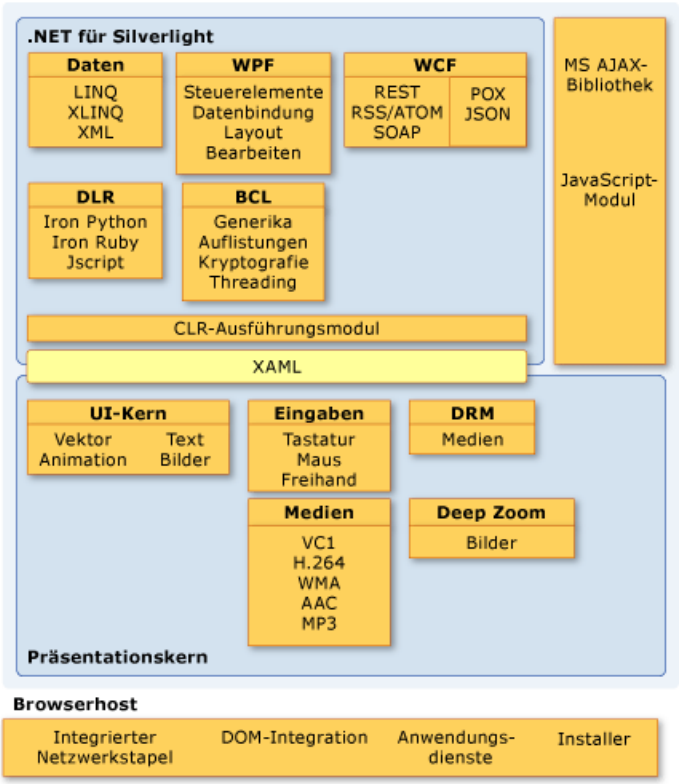


**Abbildung 21: Silverlight-Architektur [SILARCH2011]**

Das Präsentationsframework stellt jene Komponenten und Dienste zu Verfügung, mit denen die Benutzeroberfläche gestaltet und die Funktionalität der Web-Anwendung erstellt werden können:

- Die **Eingabe**-Komponente behandelt sämtliche Inputs, die ein Anwender bei der Interaktion mit der Applikation durch Tastatur, Maus usw. auslöst.

- Das **UI-Rendering** ist für das Rendern der Bitmap- und Vektorgraphiken sowie der Animationen und Texte der Benutzeroberflächen-Steuerungselemente zuständig.
- Verschiedene Typen von Audio- und Videodateien wie MP3, WMA usw. können durch die **Medien**-Komponente wiedergegeben und verwaltet werden.
- Hochauflösende Bilder werden durch **Deep-Zoom** vergrößert (Abbildung 22).
- **Steuerelemente** können durch Styles und Templates angepasst werden.
- Mit **Data-Binding** werden Steuerelemente mit Datenobjekten verknüpft, z. B.: Änderungen in einem Datenobjekt werden in einem DataGrid sofort sichtbar.
- **Layout** übernimmt die dynamische Anordnung von Elementen der Benutzeroberfläche.
- Die **DRM**-Komponente übernimmt die Verwaltung digitaler Rechte von Medien.
- Mit **XAML** (Extensible Application Markup Language) werden, wie erwähnt, die Layouts der Benutzeroberfläche erstellt. Der integrierte Parser verarbeitet die erstellten XAML-Markups weiter.


**Abbildung 22: Deep-Zoom-Effekt**[13]

In der ersten Version von Silverlight (1.0) wurden dynamische Inhalte ausschließlich durch die Verwendung von JavaScript erzeugt. Es gibt nach wie vor eine eigene MS AJAX-Bibliothek, die Silverlight-Steuerelemente für die Erstellung von Benutzeroberflächen anbietet. Diese Bibliothek wurde seit der Erweiterung der .NET-for-Silverlight-Komponente in der Version 2.0 kaum weiterentwickelt. Mit der Erweiterung wurden zusätzliche Programmiersprachen unterstützt. Neben JavaScript konnten nun auch die Skriptsprachen Python und Ruby sowie die .NET-Sprachen C# und Visual-Basic für die Entwicklung der

[13] Quelle: http://memorabilia.hardrock.com/

Programmlogik verwendet werden. Verwendet man eine der .NET-Sprachen, müssen die Quelldateien zuerst in ein .NET-Assembly[14] übersetzt werden. Diese Assembly-Dateien werden im Anschluss auf den Server geladen und können durch die Common-Language-Runtime, in der Silverlight-PlugIn integriert ist, ausgeführt werden. Folgende Funktionen und Bibliotheken werden vom .NET Framework für Silverlight angeboten:

- **Daten:** Mit der Unterstützung von LINQ (Language Integrated Query) können Abfragen in Datenbanken und XML-Files durchgeführt werden.
- **BCL:** Mit der Basisklassenbibliothek werden Programmierfunktionen zur Verfügung gestellt, wie z. B.: Zeichenfolgenbehandlung, Ein- bzw. Ausgaben, Auflistungen, reguläre Ausdrücke usw..
- **WCF:** Die Windows-Communication-Foundation ist ein Kommunikations-Framework, das den Zugriff und die Verwaltung von Remotediensten und –daten vereinfacht.
- **CLR:** Die Common-Laguage-Runtime organisiert die Speicherverwaltung, bereinigt automatisch den Speicher (Garbage-Collection), behandelt Ausnahmen (Exception-Handling) und überprüft die Typsicherheit der definierten Datentypen.
- **WPF:** Die Windows-Presentation-Foundation ist eine umfangreiche Sammlung von Steuerelementen, wie z. B.: Button, Label, ComboBox, DataGrid, usw.
- **DLR:** Die Dynamic-Language-Runtime erweitert die CLR dahingehend, dass die Ausführung von dynamischen Sprachen, wie z. B.: JavaScript, PHP, Ruby, innerhalb des .NET-Frameworks erleichtert wird. Weiters können dynamische Funktionen den Windows-eigenen statisch typisierten Sprachen hinzugefügt werden. Statisch typisierte Sprachen sind z. B.: C# und Visual Basic.[15]

Als Entwicklungsumgebung von Silverlight-Applikationen empfiehlt sich das kostenpflichtige Visual-Studio. Dieses ist seit kurzem in der Version 2010 auf dem Markt. Als Pendent zur Flash-Umgebung von Adobe bietet Microsoft das Werkzeug Expression-Blend an. Expression-Blend dient als Design- und Animationswerkzeug für WPF-basierte Anwendungen. Zusätzlich kann es als graphische Oberfläche für die Entwicklung von XAML-basierten Benutzeroberflächen verwendet werden.

---

[14] .NET-Assembly ist eine Zusammenstellung übersetzter .NET-Programmdateien

[15] setzt voraus, dass Option-Explicit-On verwendet wird. Die Option-Explicit-On-Anweisung erzwingt eine explizite Deklaration aller Variablen, wie es bei einem sauberen Programmierstil eigentlich immer der Fall sein sollte.

### 5.2.5 Oracle – JavaFX

Als Reaktion auf Flex von Adobe und Silverlight von Microsoft veröffentlichte SUN-Microsystems im Dezember 2008 JavaFX. Nachdem die Java-Applets-Technologie es nach ihrem Erscheinen nicht schaffte, sich in einem breiten Umfeld am RIA-Markt zu etablieren, versuchte SUN-Microsystems und, nach deren Übernahme, ORACLE, den Anschluss an die Konkurrenz herzustellen. Eines der Hauptprobleme der Applets waren die verhältnismäßig großen Ressourcen-Anforderungen. Diese haben sich bei gegenwärtigen Rechnerkapazitäten relativiert. Die Philosophie von JavaFX weicht deshalb prinzipiell nicht stark von dem durchdachten Aufbau der Java-Applets ab. Es galt aber, die umständliche und eingeschränkte Gestaltung der Benutzeroberflächen mit Swing und AWT durch eine neue Technologie zu ersetzen [vgl. STEYER2009].

Mit JavaFX wurde den Entwicklern eine neue Möglichkeit für die Gestaltung der graphischen Oberflächen gegeben, womit auch bei SUN/ORACLE die Trennung von Design und Programmlogik vollzogen wurde. Zu den Konkurrenzprodukten unterscheidet sich JavaFX erheblich, denn es wurde nicht auf einem XML-Dialekt aufgesetzt, dessen Vorzüge schon an den zuvor beschriebenen Technologien erläutert wurde, sondern es wurde eine eigene Skript-Sprache entwickelt, die eher eine Verwandtschaft zu JavaScript aufweist. Diese als JavaFX-Script bezeichnete Sprache ist nicht nur in der Lage, GUI-Komponenten zu definieren, sondern ermöglicht es einen Entwickler auch, Programmlogik in Form von Methoden bzw. Events in den Programmcode zu integrieren. Der letzte Punkt relativiert natürlich das Ansinnen Design und Programmlogik strikt zu trennen, außer es wird der Logik-Code nur für graphische Veränderungen innerhalb der Benutzeroberfläche verwendet. Der Script-Code für die Erstellung der graphischen Komponenten dient zur Instanziierung der Swing-Objekte und deren Wertbelegung, vergleichbar mit XAML und den WPF-Komponenten.

Durch die Nutzung des Java-Runtime-Environment (JRE) bzw. des Java-Development-Kit (JDK) kann bei der Entwicklung von JavaFX-Applikationen auf sämtliche Java-Bibliotheken zurückgegriffen werden. Das JRE ist auch die einzige Voraussetzung, um JavaFX-Inhalte über einen Browser zu nutzen, d. h. wenn ein Java-Runtime-Environment auf einem Desktop-PC oder einem Mobile-Device installiert ist, sind keine weiteren Browser-PlugIns von Nöten. Durch die weite Verbreitung des JRE und durch den Umstand, dass für nahezu alle Plattformen ein passendes JRE zur Verfügung steht, ergibt sich eine hohe Plattformunabhängigkeit, sowohl für die Nutzer als auch für die Entwickler einer Web-Anwendung. Um Offline mit JavaFX-Web-Applikationen zu arbeiten, ist es seit der Version 6

update 10 von Java-SE möglich, Applikationen einfach aus dem Browser per Drag-and-Drop auf dem Desktop zu installieren.

Aufbauend auf die Java-Virtual-Machine besteht die JavaFX-Plattform-Architektur [JAVAFX2011] aus Entwickler-Tools für die Ersteller der Programmlogik und Designer-Tools für die Oberflächengestaltung (Abbildung 23):

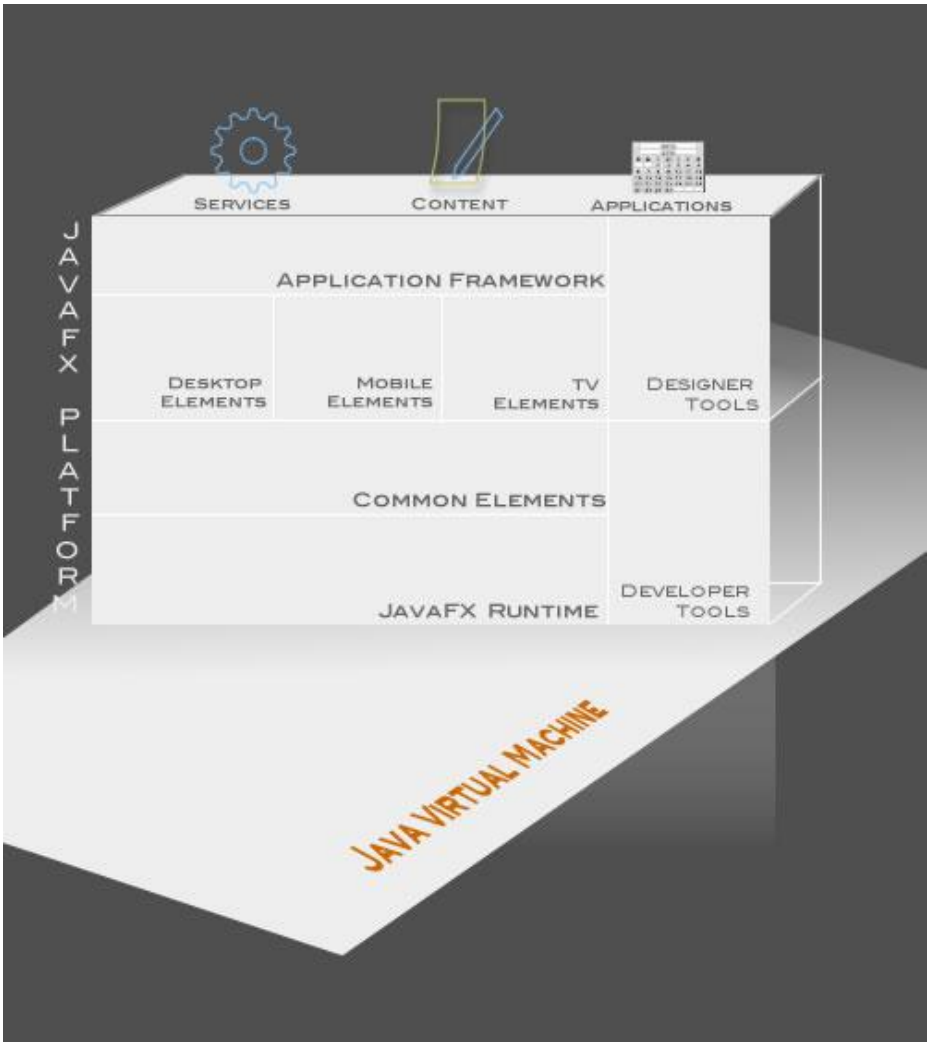


**Abbildung 23: JavaFX Plattform-Architektur [JAVAFX2011]**

Die Entwicklungswerkzeuge beinhalten die Komponenten:

- **JavaFX-Runtime** ist die spezifische Laufzeitumgebung von JavaFX und wird automatisch mit dem Java-Runtime-Environment installiert.
- **Common-Elements** beinhaltet APIs und weitere Laufzeitumgebungen für die plattformübergreifende Zusammenarbeit.

Für das Designen der Oberflächen stehen folgende Elemente zur Verfügung:

- Die **Desktop-Elements** bieten APIs zur Oberflächengestaltung von Desktopanwendungen, egal ob sie direkt am Desktop oder in einem Browser angezeigt werden.
- **Mobile-Elements** werden für die Gestaltung der Benutzeroberflächen von Mobile-Devices verwendet.
- **TV-Elements** stellt APIs zur Verfügung mit denen über Set-top-Boxen Inhalte auf digitalen TV-Bildschirmen angezeigt werden können.
- Im **Application-Framework** werden die graphischen Komponenten zur Verfügung gestellt, die bei der Oberflächengestaltung instanziiert werden können.

Als Entwicklungsumgebung bietet ORACLE ein Developer-Bundle und ein Designer-Bundle zum freien Download an. Das Developer-Bundle richtet sich an die Programm-Logik-Ersteller und besteht aus der altbekannten NetBeans-IDE, in die wiederum das JavaFX-Plugin-for-Netbeans und die JavaFX-SDK integriert sind. Das FX-Plugin kann auch jederzeit in eine schon bestehende NetBeans-Installation integriert werden.

Das Designer-Bundle beinhaltet die JavaFX-Production-Suite und kann von Oberflächengestaltern als Gesamtpaket ebenfalls frei heruntergeladen werden. Es beinhaltet die JavaFX-PlugIns für Adobe-Photoshop und Adobe-Illustrator. Durch die PlugIns können Graphiken direkt aus Photoshop und Illustrator in das JavaFX-Format exportiert werden. Ebenso beinhaltet die JavaFX-Production-Suite die JavaFX-Media-Factory. Diese besteht aus dem SVG-Converter, um SVG-Graphiken in das JavaFX-Format zu konvertieren, und dem JavaFX-Graphics-Viewer, mit dem Graphiken, die in das JavaFX-Format konvertiert wurden, betrachtet werden können.

Um JavaFX-Applikationen auch in anderen Entwicklungsumgebungen (z.B.: Eclipse) zu erstellen, kann die JavaFX-SDK auch als stand-alone-SDK heruntergeladen werden.

## 5.3 Zusammenfassung und Gegenüberstellung

Anhand der bisherigen Betrachtungen der RIA-Technologien ist es offensichtlich, dass jede Technologie Vor- und Nachteile in sich birgt und keines der Produkte den Anderen in jeder Hinsicht überlegen ist. In der anschließenden Tabelle werden die in Frage kommenden RIA-Technologien überblicksmäßig anhand eines Kriterienkatalogs, der im Besonderen auf den Hauptanspruch des Metamodell-Browsers (visuell ansprechende Darstellung von Metamodellen) ausgelegt ist,  bewertet.

| | Ajax/XHTML | Flex/Flash | Silverlight | JavaFX |
|---|---|---|---|---|
| **Anwender-erfahrung (User Experience)** | **Genügend** | **Sehr gut** | **Sehr gut** | **Gut** |
| | Die Anwendererfahrung beschreibt jenen Eindruck eines Anwenders, der bei der Verwendung einer Web-Applikation vermittelt wird. Dieser Eindruck ist auf jeden Fall immer subjektiv. Im Fall des Metamodell-Browsers wird davon ausgegangen, dass der Anwender eine moderne Web 2.0-Anwendung als positiv empfindet, die z. B.: Transparenzen und diverse Effekte inkludiert. Ajax/XHTML ist bei der Darstellung von Inhalten stark durch die Browservorgaben eingeschränkt. Die anderen Technologien bieten weit mehr Möglichkeiten bei der Darstellung von Webinhalten und Effekten. | | | |
| **Anwender-aufwand** | **Sehr gut** | **Genügend** | **Genügend** | **Befriedigend** |
| | Bei Ajax/XHTML-Applikationen müssen durch den Anwender keine Updates durchgeführt werden, weil die ausführende Technologie in den Browsern integriert ist. Bei den anderen Technologien müssen regelmäßig Updates durchgeführt werden. | | | |
| **Verbreitung** | **100%** | **Ca. 97%** | **Ca. 60%** | **Ca. 70%** |
| **Browser-kompatibilität** | **Befriedigend** | **Sehr gut** | **Sehr gut** | **Sehr gut** |
| | Durch Browserinkompatibilitäten werden Ajax/XHTML-Applikationen in unterschiedlichen Browsern oft auch unterschiedlich dargestellt. Die Plug-Ins der anderen Technologien garantieren in jedem Browser die gleiche Darstellung. | | | |
| **Plattform-unabhängigkeit** | **Gut** | **Sehr gut** | **Genügend** | **Sehr gut** |
| **Entwicklungs-unterstützung** | **Genügend** | **Sehr gut** | **Gut** | **Befriedigend** |
| | Für Ajax werden freie Komponenten-Bibliotheken angeboten, wobei die Implementierung auf verschiedenen Plattformen immer wieder zu Problemen führt. Für Flex-Anwendungen gibt es sehr gute Designunterstützung durch die Adobe-Creative-Suite. Ähnliches gilt für Silverlight mit dem Expression-Studio. Für JavaFX existieren Plug-Ins, um Graphiken aus Photoshop und Illustrator in das JavaFX-Format zu exportieren. | | | |
| **SEO / Erreichbarkeit durch Suchmaschinen** | **Sehr gut** | **Genügend** | **Genügend** | **Befriedigend** |
| | Inhalte von Ajax-Anwendungen sind am Einfachsten für Suchmaschinen zu indizieren, da diese auf lesbaren HTML basieren. Die anderen Technologien müssen alternativ Meta-Daten zur Verfügung stellen, welche von den Suchmaschinen indiziert werden können. JavaFX bietet für eine bessere Erreichbarkeit Screen-Reader an. | | | |

| | | | | |
|---|---|---|---|---|
| **Graphische Darstellung** | **Genügend** | **Sehr gut** | **Sehr gut** | **Gut** |
| | Im Gegensatz zu den anderen Technologien ist Ajax bei der Darstellung von graphischen Elementen Pixel-basiert und nicht Vektor-basiert. Dadurch ergeben sich erhebliche Nachteile beim Skalieren und bei unterschiedlichen Auflösungen. | | | |
| **User-Interface Gestaltung** | **Genügend** | **Sehr gut** | **Sehr gut** | **Befriedigend** |
| | Für Ajax-Anwendungen existieren nur verhältnismäßig bescheidene Komponenten-Bibliotheken. Bei den restlichen Technologien gibt es kaum Einschränkungen bei der Erstellung von eigenen Komponenten, Effekten und Animationen. JavaFX verfügt momentan aber noch nicht über geeignete Entwicklungswerkzeuge. | | | |
| **Kosten** | **Gut** | **Genügend** | **Genügend** | **Sehr gut** |
| | Ajax verursacht zusätzliche Kosten durch die Anpassung an die verschiedenen Browser. Ansonsten stehen sowohl für Ajax als auch für JavaFX Entwicklungsumgebungen zur freien Verfügung. Die Entwicklungsumgebungen für Flex und Silverlight sind hingegen kostenpflichtig. | | | |
| **Fazit** | **2.8** | **2.0** | **2.6** | **2.2** |

**Tabelle 1: RIA-Technologien Bewertung**

Für den Metamodell-Browser besonders relevante Punkte werden in Folge gesondert diskutiert.

### 5.3.1 Browserkompatibilität

Reine **AJAX/(X)HTML**-Applikationen und zahlreiche weitere Anwendungen, die mit dem Google-Web-Toolkit erstellt wurden, zeichnet eine sehr gute Browserkompatibilität aus. Die verwendeten Technologien für die Darstellung der Web-Inhalte setzen weder ein Browser-PlugIn noch die Installation einer Laufzeitumgebung auf dem Client-Rechner voraus. Das **Google-Web-Toolkit** ist in dieser Hinsicht noch eine Stufe über eine reine AJAX/(X)HTML-Anwendung zu stellen, weil es für die meisten handelsüblichen Browser eine eigene Permutation der Applikation generiert. Dadurch, dass das Google-Web-Toolkit auf browserspezifische Eigenheiten bei der Interpretation des Quellcodes explizit Rücksicht nimmt, kann eine umfassende Browserkompatibilität gewährleistet werden. **Flex** und **Silverlight** setzen die Installation eigener Browser-PlugIns voraus. Dieser Umstand könnte dann problematisch werden, wenn einem potentiellen Anwender der Web-Applikation aus firmenpolitischen Gründen oder aus eigenem Unvermögen eine Installation des betreffenden

PlugIns verwehrt bleibt. Ähnliches gilt auch für das Java-Runtime-Environment, dessen Installation am Client-Rechner die Voraussetzung für die Darstellung von JavaFX-Inhalten darstellt.

### 5.3.2 Plattformunabhängigkeit

In Bezug auf die Plattformunabhängigkeit gibt es für **AJAX/(X)HTML**[16]-Anwendungen keinerlei Einschränkungen, sofern auf dieser Plattform ein W3C-standardkonformer Browser installiert werden kann. Ähnliches gilt für **Flex** von Adobe. Sofern es ein Flashplayer-PlugIn für den Browser gibt, der auf der betreffenden Plattform installiert wurde, können Flex-Anwendungen auf der jeweiligen Plattform auch ausgeführt werden. Das gleiche gilt für **JavaFX**-Applikationen. Sofern die Plattform das Java-Runtime-Environment unterstützt, gibt es keinerlei Einschränkungen bei der Ausführung von JavaFX-basierten Anwendungen. Gänzlich anders verhält es sich bei Microsoft-**Silverlight**. Silverlight ist nur auf jenen Plattformen lauffähig, die von Microsoft unterstützt werden, und das sind bis dato nur die Microsoft-Betriebssysteme, welche das .NET-Framework ab Version 3.0 unterstützen sowie Apple-Macintosh, hier jedoch nur die Browser Internet-Explorer, Firefox und Safari. Von Novell wird das Moonlight-PlugIn angeboten, mit dem Silverlight-Inhalte auch für Linux-Anwender zugänglich werden. Solche Open-Source-Lösungen sind in den meisten Fällen aber immer mit Einschränkungen verbunden.

### 5.3.3 Entwicklung

Für den Entwickler einer Web-Applikation ist bei der Wahl der Technologie das eigene Vorwissen von entscheidender Bedeutung. Für die Entwicklung von reinen **AJAX/(X)HTML-Anwendungen** ist das Erlernen verschiedenster Sub-Technologien erforderlich. AJAX-Anwendungen sind ein Konglomerat aus HTML, CSS, JavaScript und JSON. Für die Gestaltung von RIA-Benutzeroberflächen gesellt sich als zusätzliche Anforderung das Wissen von JavaScript-Frameworks wie Jquery, ExtJS und Dojo hinzu. Speziell JavaScript setzt eine gewisse Erfahrung des Entwicklers voraus und ist immer wieder für eine Überraschung gut, besonders für Entwickler, die hauptsächlich mit objektorientierten Sprachen zu tun hatten.

[16] Dadurch, dass das Google-Web-Toolkit im Endeffekt aus Java-Code eine AJAX/(X)HTML-Anwendung generiert, wird auf dieses Tool nur noch eingegangen, wenn es explizit sinnvoll erscheint.

Das **Google-Web-Toolkit** birgt den großen Vorteil, dass Web-Anwendungen rein mit Java-Code programmiert werden können. Die Übersetzung in AJAX/(H)TML übernimmt im Anschluss der GWT-Compiler. Für Entwickler, die mit Java-Programmierung schon Erfahrung gesammelt haben und speziell Erfahrung bei der Erstellung von Benutzeroberflächen mit Swing oder AWT besitzen, sollte es ein leichtes sein, funktional ansprechende RIA-Anwendungen zu erstellen. Visuell ansprechende Applikationen sind hingegen aufgrund des gewaltigen Programmieraufwands mit anderen Technologien leichter zu bewerkstelligen.

Mit **JavaFX** kann bei der Programmierung der Anwendungslogik ebenfalls auf Java zurückgegriffen werden. Somit kann ein Entwickler, wenn er schon Erfahrung mit Java-basierten Desktop-Applikationen innehat, auf ein umfassendes Vorwissen zurückgreifen. Schwieriger gestaltet sich die Gestaltung der Benutzeroberflächen, die mit JavaFX-Script umgesetzt werden müssen. Da es sich nicht um einen XML-Dialekt handelt wie bei Silverlight oder Flex, und damit weniger intuitiv ist, setzt dies einen größeren Einarbeitungsaufwand voraus.

**Flex** setzt bei der Gestaltung der Oberflächen auf den XML-Dialekt MXML. Dieser sollte jedem Entwickler, der der Markup-Sprache XML mächtig ist, einen leichten Zugang zum Designen einer Benutzeroberfläche bieten. Für die clientseitige Programmlogik verwendet Flex ActionScript. ActionScript entspricht zwar keiner gängigen Programmiersprache, folgt aber den Grundsätzen der Objektorientierung. Damit sollte die Script-Sprache nach einer kurzen Einarbeitungszeit keine größere Hürde für einen Entwickler darstellen. Serverseitig kann bei Flex durch das BlazeDS-Framework für die Programmlogik auf Java zurückgegriffen werden.

Zu diesem Aspekt bietet **Silverlight** klare Vorteile. Auf der einen Seite kann zur graphischen Gestaltung auf den XML-Dialekt XAML zurückgegriffen werden, andererseits kann sich der Entwickler für die Erstellung der Programmlogik eine vom .NET-Framework unterstützte Sprache aussuchen, z. B.: C# oder Visual-Basic.

### 5.3.4 Gestaltungsmöglichkeiten

Bei den Möglichkeiten der Oberflächengestaltung von Web-Anwendungen bestehen erhebliche Unterschiede zwischen den einzelnen Technologien. Rein technisch wäre es möglich, jede visuelle Anforderung an eine Web-Applikation auch auf Basis der **AJAX/(X)HTML**-Technologie umzusetzen. In der Praxis würde dieses Ansinnen einen

erheblichen Entwicklungsaufwand voraussetzen und bei multimedialen Inhalten würde die JavaScript Performance  sehr schnell an ihre Grenzen stoßen. Jede einzelne Komponente, die nicht Teil einer JavaScript-Bibliothek ist, müsste mühsam per Hand neu programmiert werden, visuelle Effekte würden die Geschwindigkeit weiter beeinträchtigen.

Im Gegensatz dazu bieten **Flex**, **Silverlight** und **JavaFX** umfangreiche Komponenten-Bibliotheken, wobei die einzelnen Komponenten individuell angepasst werden können. Auch die Erstellung neuer Komponenten und die Einbindung externer Medien werden umfangreich unterstützt. Ebenso können visuell ansprechende Animationen auf recht einfache Weise in die Applikation integriert werden.

Reichen die integrierten Funktionen von Flex und Silverlight für die Darstellung animierter Inhalte noch nicht aus, so bietet Adobe mit Flash und Microsoft mit Expression-Blend Entwicklungswerkzeuge an, mit denen jede technisch machbare Animation erstellt und anschließend in die jeweilige Umgebung (Flex bzw. Silverlight) integriert werden kann. Mit JavaFX ist es zwar auch möglich, ansprechende Animationen zu erzeugen, nur erfordert dies deutlich mehr Programmieraufwand, da es vorerst kein Animations-Entwicklungswerkzeug spezifisch für JavaFX-Applikationen gibt. Immerhin bietet die Version 1.3 von JavaFX die Möglichkeit, Graphiken aus Photoshop und Illustrator in das JavaFX-Format zu exportieren.

## 5.4 Bewertung

Anhand der betrachteten Technologien gilt es nun, eine RIA-Lösung auszuwählen, die den Anforderungen des Projektinhalts (Metamodell-Browser) und der Adressatengruppe (Open-Model-Initiative) in optimaler Weise entspricht.

Da es sich beim Metamodell-Browser um eine Applikation handelt, die möglichst vielen Mitgliedern der Open-Model-Community offen stehen soll, fällt Silverlight wegen dem verhältnismäßig geringsten Verbreitungsgrad (siehe Abbildung 24: Microsoft-Silverlight bewegt sich bei einem Verbreitungsgrad von ca. 60%)[17] und der Plattformabhängigkeit als Technologie für den Metamodell-Browser aus. Speziell bei der Plattformabhängigkeit gilt zu bedenken, dass Mitglieder der Open-Model-Community nicht als Mainstream-User zu erwarten sind und es durchaus vorstellbar ist, dass viele Community-Mitglieder Alternativ-Plattformen benutzen.

[17] An dieser Stelle sei erwähnt, dass sich diese Quote innerhalb eines Jahres von 30% auf ca. 60% verdoppelt hat.

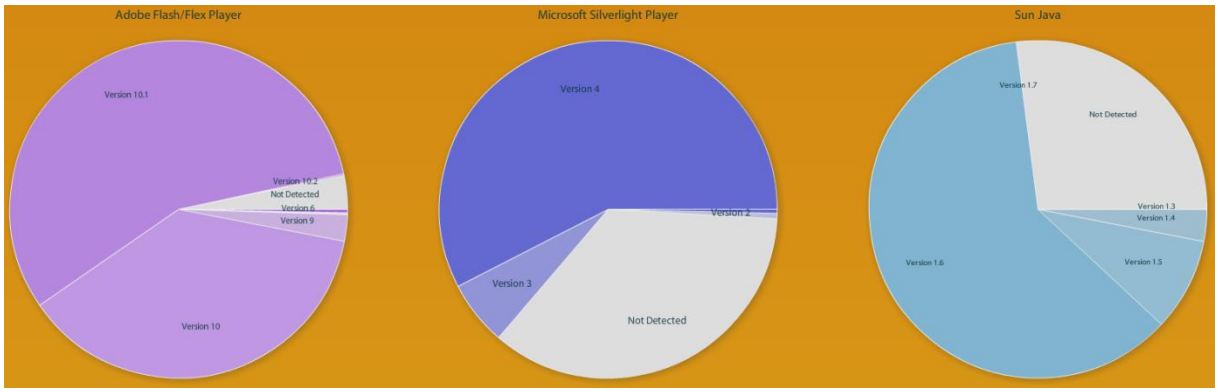


**Abbildung 24: Verbreitung der RIA-Technologien**[18]

Durch die Anforderung an eine „visuell ansprechende“ Web-Applikation wird auch eine traditionelle AJAX/(X)HTML-Variante nicht ins nähere Kalkül gezogen. Zu groß wäre der Aufwand, komplexe Komponenten zu erstellen und ansprechende Animationen mit JavaScript umzusetzen. Dadurch bietet auch das Google-Web-Toolkit keine entsprechende Alternative.

JavaFX ist durch das Java-Runtime-Environment in hohem Grade plattformunabhängig und genießt zusätzlich, oder gerade deswegen, eine große Verbreitung. Dazu wurde die Performance von JavaFX-Applikationen seit Version 1.3 erheblich verbessert. Die Möglichkeit, Oberflächen ansprechend zu gestalten, ist durchaus gegeben. Nur steht kein Entwicklungswerkzeug für die Gestaltung der Animationen zur Verfügung, wodurch die Entwicklung sehr aufwändig wäre.

Nach diesem Ausschlussverfahren fällt somit die Wahl für die zu verwendende Technologie zur Erstellung des **Metamodell-Browser-Prototyps** auf Flex von Adobe. Einerseits besitzt der FlashPlayer einen nahezu flächendeckenden Verbreitungsgrad, andererseits bietet Flex umfassende Möglichkeiten bei der Gestaltung einer visuell ansprechenden Benutzeroberfläche. Sollte Flex an seine Grenzen stoßen, so liefert Flash als Animationswerkzeug noch zusätzliche Möglichkeiten der Realisierung. Durch die Verwendungsmöglichkeit von Java im Server-Backend sollte es auch keine Probleme geben, die relevanten Daten über Web-Services aus Metamodell-Repositories abzurufen und für die Darstellung im Metamodell-Browser aufzubereiten.

[18] http://www.riastats.com(Stand:18.10.2010)

# 6 Konzeptionelle und prototypische Implementierung

## 6.1 Anforderungsanalyse

Im folgenden Kapitel werden die grundsätzlichen Entwurfsvoraussetzungen für den Metamodel-Browser beschrieben. Danach werden die Anforderungen an den zu entwickelnden Prototypen erhoben. Sowohl die Entwurfsvoraussetzungen als auch die Anforderungen an den Prototyp resultieren aus der Bedarfsanforderung der Open-Model-Initiative, Gesprächen mit Chef- und Co-Entwicklern der Open-Model-Community, den Diskussionen innerhalb zweier Open-Model-Workshops sowie eigenen Überlegungen.

### 6.1.1 Entwurfsvoraussetzungen (Even)

- **EV1 – Daten im XML-Format**:
  Die Daten der Metamodelle sind im XML-Format gespeichert. Dieser Umstand resultiert daraus, dass die durch den Prototypen des Metamodel-Browsers zu visualisierenden Metamodelle aus der ADOxx-Plattform (v1.1) als XML-Files exportiert werden können.
- **EV2 – Rich Internet Applikation:**
  Der Metamodell-Browser dient den Mitgliedern der Open-Model-Community als Informationsquelle über bestehende Metamodelle. Deshalb wird der Browser zur Steigerung der Akzeptanz und der Benutzerfreundlichkeit als Rich-Internet-Applikation implementiert. Somit sind keine Installation eines speziellen Plug-Ins sowie keine Registrierung bei einem Web-Service durch den User erforderlich. Zusätzlich werden die Vorteile einer reinen Webanwendung mit dem „Look and Feel“ einer herkömmlichen Desktopanwendung kombiniert.
- **EV3 – Adobe Flex:**
  Aufgrund der Evaluation in Kapitel 5.3 wurde die Flex-Technologie von Adobe für die Implementierung des Prototypen auserkoren. Durch die weite Verbreitung des Flash-Players, die nahezu uneingeschränkte Browserkompatibilität und die hervorragenden Möglichkeiten zur grafischen Darstellung eignet sich Flex am besten als RIA-Technologie. Einschränkungen. Die bei der Benutzung durch Mobile-Devices entstehen können, werden wegen mangelnder Relevanz nicht berücksichtigt.

- **EV4 – Serverseitige Java-Logik:**
  Die Verwendung von serverseitiger Java-Logik ermöglicht Entwicklern (auch ohne Flex-Kenntnisse), Erweiterungen zu integrieren, die den Zugriff auf zusätzliche Metamodell-Datenquellen (z. B.: über JSON, Hibernate,…) ermöglichen. Generell bietet die Verwendung von Java eine gute Basis für den funktionellen Ausbau des Prototyps.
- **EV5-Tomcat:**
  Die Verwendung eines Apache-Tomcat als Application-Server entspricht der Bedarfsanforderung der Open-Model-Initiative.

### 6.1.2 Anforderungen

Anhand der Bedarfsanforderung der Open-Model-Initiative, den zuvor erwähnten Gesprächen und eigenen Überlegungen wurden folgende Anforderungen an die Applikation definiert. Für die Darstellung des Anforderungskatalogs wurde Anleihe am *Volere Requirements Specification Template* genommen (Abbildung 25).

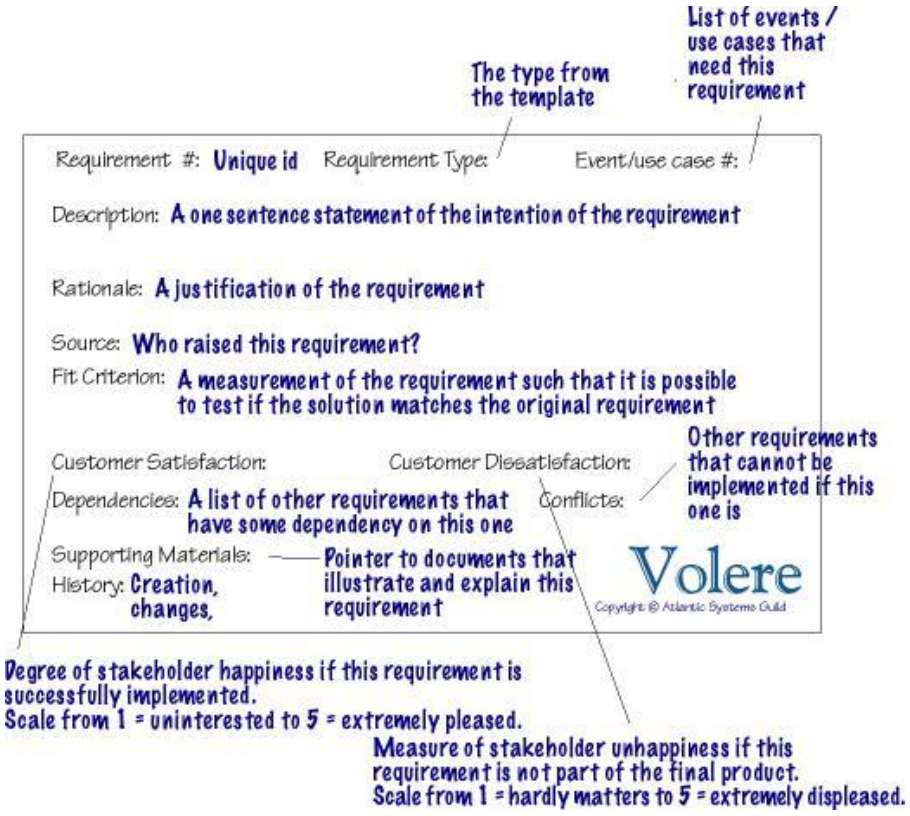


**Abbildung 25: Volere Requirement Specification Template – Requirement Shell**[19]

[19] http://www.apptius.com/Documents/VolereSRStemplate.pdf

In der folgenden Liste werden die relevanten bzw. entlehnten Punkte erläutert:

1. **Requirement (Anforderungs-ID):** Eindeutige Identifikationsnummer
2. **Requirement Type (Anforderungstyp):** Anforderungstyp, der im *Volere Requirement Specification Template* über eine eindeutige ID zugeordnet ist
3. **Description (Beschreibung):** Beschreibung, die in einem Satz die Intention der Anforderung zusammenfasst
4. **Rationale (Begründung):** Begründung für die Anforderung
5. **Fit Criterion (Erfüllungskriterium):** Kriterium, mit dem überprüft wird, ob die Anforderung erfüllt wurde
6. **Dependencies (Abhängigkeiten):** Abhängigkeit zu anderen Anforderungen

Add 2: Im Volere Requirement Specification Template sind 27 Anforderungstypen definiert, die zum Teil für dieses Projekt von nur geringer Bedeutung sind (z.B.: 16. Kulturelle und politische Anforderungen), weshalb in erster Linie folgende Typen relevant sind:

9. **Functional and Data-Requirements** (Funktionale- und Datenanforderungen)
   Welche Aktionen sollen mit der Applikation durchgeführt werden können
10. **Look and Feel-Requirements** („Look and Feel“ Anforderungen)
    Anforderungen an das Design und die Bedienbarkeit
11. **Usability-Requirements** (Benutzerfreundlichkeits-Anforderungen)
    Welches Vorwissen muss ein User mitbringen, um die Applikation zu nutzen
12. **Performance-Requirements** (Leistungsanforderungen)
    Anforderungen für die Geschwindigkeit und Zuverlässigkeit der Applikation
13. **Operational-Requirements** (Anforderungen an den Einsatzbereich)
    Auf welchen Plattformen wird die Applikation zum Einsatz kommen
14. **Maintainability and Portability Requirements** (Anforderungen an Wartung und Übertragbarkeit)
    Inwieweit ist die Applikation Erweiterbar und auf andere Plattformen übertragbar

### 6.1.3 Anforderungen an den Metamodel-Browser

| **Anforderungs-ID:** | 1 | **Anforderungstyp:** | 10 |
|---|---|---|---|
| **Beschreibung:** | Der Metamodel-Browser soll eine visuell ansprechende Benutzeroberfläche bieten | | |
| **Hintergrund:** | Durch eine visuell ansprechende Benutzeroberfläche soll der User zur Interaktion mit dem Metamodel-Browser angeregt werden. Ein „Wow“-Effekt soll bei neuen Mitgliedern der **Open-Model-Community** Interesse wecken, auch wenn sie mit den dargebotenen Inhalten unter Umständen überfordert sind. | | |
| **Erfüllungskriterium:** | Die Benutzeroberfläche wurde gemäß der Entwurfsvoraussetzung 3 mit Adobe Flex gestaltet. Verschiedene Effekte verstärken die „User-Experience“ | | |
| **Abhängigkeit:** | | | |

**Tabelle 2: Anforderung 1 (visuell ansprechende Benutzeroberfläche)**

| **Anforderungs-ID:** | 2 | **Anforderungstyp:** | 11 |
|---|---|---|---|
| **Beschreibung:** | An dominanter Stelle soll ein User-Guide die grundsätzliche Funktionalität der Applikation erklären. | | |
| **Hintergrund:** | Erfahrenen Usern soll ein schneller Einstieg in die Applikation möglich sein.<br>Benutzer mit wenigen Vorkenntnissen soll die Interaktion mit dem Metamodel-Browser ermöglicht werden. | | |
| **Erfüllungskriterium:** | Ein User-Guide wurde der Applikation hinzugefügt | | |
| **Abhängigkeit:** | 1 | | |

**Tabelle 3: Anforderung 2 (User Guide)**

| **Anforderungs-ID:** | 3 | **Anforderungstyp:** | 10 |
|---|---|---|---|
| **Beschreibung:** | Die Wartezeit, bis die Applikation geladen wird, soll durch die Integration eines *Preloaders* „verkürzt“ werden. | | |
| **Hintergrund:** | Durch die Umsetzung der Applikation mit Flex-Technologie kann es beim Laden des Metamodel-Browsers zu Verzögerungen kommen. Der User soll dabei nicht vor einem weißen Bildschirm sitzen. | | |
| **Erfüllungskriterium:** | Eine Preloader-Komponente wurde entwickelt und in die Applikation integriert. | | |
| **Abhängigkeit:** | 1 | | |

**Tabelle 4: Anforderung 3 (Preloader implementieren)**

| | | | |
|---|---|---|---|
| **Anforderungs-ID:** | 4 | **Anforderungstyp:** | 12 |
| **Beschreibung:** | Eine möglichst hohe Performance bzw. ein schneller Datenabruf soll geboten werden. | | |
| **Hintergrund:** | Um eine gute Usability zu gewährleisten, ist es wichtig, dass der Datenaustausch so schnell wie möglich erfolgt. | | |
| **Erfüllungskriterium:** | Um die Daten aus den XML-Files für den Metamodel-Browser aufzubereiten, wurde auf dem Server eine Java-Logik implementiert. Mit Hilfe der Blaze-DS Technologie werden die zur Laufzeit instanziierten Java-Objekte direkt in ActionScript-Objekte transformiert, ohne den Umweg über Serialisierung und Deserialisierung zu gehen. | | |
| **Abhängigkeit:** | | | |

**Tabelle 5: Anforderung 4 (Gute Performance)**

| | | | |
|---|---|---|---|
| **Anforderungs-ID:** | 5 | **Anforderungstyp:** | 9 |
| **Beschreibung:** | Der Metamodel-Browser soll Metamodelle, die im XML-Format gespeichert sind, visualisieren. | | |
| **Hintergrund:** | Die Metamodelle, die im Prototyp des Metamodell-Browsers visualisiert werden sollen, werden im XML-Format aus der ADOxx Plattform (v1.1) exportiert. | | |
| **Erfüllungskriterium:** | Die serverseitige Java-Logik liest die auf dem Server hinterlegten Daten aus den XML-Files aus und transformiert die Elemente für die weitere Verarbeitung in Java-Objekte. | | |
| **Abhängigkeit:** | 4 | | |

**Tabelle 6: Anforderung 5 (Metamodelle aus dem XML-Format visualisieren)**

| **Anforderungs-ID:** | 6 | **Anforderungstyp:** | 13 |
|---|---|---|---|
| **Beschreibung:** | Der Metamodell-Browser soll auf möglichst vielen Plattformen verwendbar sein. | | |
| **Hintergrund:** | Um eine möglichst hohe Verbreitung zu gewährleisten, soll der Metamodell-Browser unabhängig von Betriebssystemen und Browsertechnologien verwendbar sein. Mobile-Devices spielen bei diesem Aspekt nur eine untergeordnete Rolle, weil diese für wissenschaftliches Arbeiten nicht von großer Relevanz sind. | | |
| **Erfüllungskriterium:** | Das Frontend des Metamodell-Browsers wurde mit der Flex-Technologie umgesetzt, die als einzige Voraussetzung das FlashPlayer-PlugIn vorgibt, welches für die verschiedensten Plattformen und Browser zur Verfügung steht. | | |
| **Abhängigkeit:** | | | |

**Tabelle 7: Anforderung 6 (plattformunabhängig)**

| **Anforderungs-ID:** | 7 | **Anforderungstyp:** | 14 |
|---|---|---|---|
| **Beschreibung:** | Die Applikation soll in Zukunft auch auf andere Metamodell-Repositorien zugreifen können | | |
| **Hintergrund:** | In Zukunft ist es denkbar, dass zusätzlich zur ADOxx-Plattform auch andere Metamodell-Repositorien als Datenquelle dienen. Dies setzt voraus, dass die Applikation insofern erweiterbar ist, sodass sie auch Metamodelle aus anderen Datenformaten verarbeiten kann. | | |
| **Erfüllungskriterium:** | Die serverseitige Programmlogik wurde in Java programmiert und eröffnet somit weitreichende Erweiterungsmöglichkeiten für andere Datenformate bzw. Datenbankzugriffe. | | |
| **Abhängigkeit:** | 4 | | |

**Tabelle 8: Anforderung 7 (Zukünftige Erweiterungen auf andere Repositorien)**

| **Anforderungs-ID:** | 8 | **Anforderungstyp:** | |
|---|---|---|---|
| **Beschreibung:** | Der Metamodell-Browser soll auf einem Apache-Tomcat-Server laufen. | | |
| **Hintergrund:** | Laut Vorgabe der Open-Model-Initiative (Entwurfsvoraussetzung 5) soll die Applikation auf einem Tomcat-Server deployed werden. | | |
| **Erfüllungskriterium:** | Ein Apache-Tomcat-Server wurde installiert und konfiguriert. | | |
| **Abhängigkeit:** | | | |

**Tabelle 9: Anforderung 8 (Apache-Tomcat-Server)**

| | | | |
|---|---|---|---|
| **Anforderungs-ID:** | 9 | **Anforderungstyp:** | 9 |
| **Beschreibung:** | Alle verfügbaren Metamodelle sollen angezeigt werden. | | |
| **Hintergrund:** | Um dem User eine Übersicht der zur Verfügung stehenden Metamodelle zu gewährleisten, müssen diese bei Start der Applikation angezeigt werden. | | |
| **Erfüllungskriterium:** | Die Bezeichnungen der zur Verfügung stehenden Metamodelle wurden korrekt ausgelesen und werden innerhalb der Applikation aufgelistet. | | |
| **Abhängigkeit:** | 1, 4 | | |

**Tabelle 10: Anforderung 9 (Anzeigen der verfügbaren Metamodelle)**

| | | | |
|---|---|---|---|
| **Anforderungs-ID:** | 10 | **Anforderungstyp:** | 10 |
| **Beschreibung:** | Nach der Wahl eines Metamodells sollen dessen Komponenten in graphischer Form angezeigt werden. | | |
| **Hintergrund:** | Damit der Modellierungs-Aspekt besser zur Geltung kommt, sollen die einzelnen Elemente als grafische Komponenten dargestellt werden. Zusätzlich soll man zwischen abstrakten und konkreten Elementen bzw. Relationen unterscheiden können. | | |
| **Erfüllungskriterium:** | Es wurden sowohl für Elemente als auch für die Relationen eigene Komponenten erstellt. Zusätzlich wurden die Elemente aufgrund von Konventionen[20] in abstrakte und konkrete Elemente unterteilt. | | |
| **Abhängigkeit:** | 9 | | |

**Tabelle 11: Anforderung 10 (Komponenten grafisch anzeigen)**

[20] Die Unterscheidung zwischen abstrakten- und konkreten Elementen erfolgt anhand der Konvention, dass abstrakte Elemente bei der Elementbezeichnung immer einen __ (Doppel-Unterstrich) vorgestellt haben.

| **Anforderungs-ID:** | 11 | **Anforderungstyp:** | 9 |
|---|---|---|---|
| **Beschreibung:** | Elemente und Relationen sollen übersichtlich angezeigt werden. | | |
| **Hintergrund:** | Um zu verhindern, dass zu viel Information auf einer Seite angezeigt wird, und um die Übersichtlichkeit durch auf- und ab scrollen nicht einzuschränken, sollen Elemente und Relationen voneinander getrennt angezeigt werden. | | |
| **Erfüllungskriterium:** | Es wurde eine getrennte Ansicht von Elementen und Relationen implementiert. | | |
| **Abhängigkeit:** | 10 | | |

**Tabelle 12: Anforderung 11 (Übersichtliche Anordnung der Komponenten)**

| **Anforderungs-ID:** | 12 | **Anforderungstyp:** | 9 |
|---|---|---|---|
| **Beschreibung:** | Per Maus-Click auf eine Komponente sollen deren Attribute angezeigt werden. | | |
| **Hintergrund:** | Für eine effektive Nutzung des Metamodell-Browsers muss der User die Möglichkeit haben, sich die Attribute jeder Komponente anzeigen zu lassen. | | |
| **Erfüllungskriterium:** | Durch einen Maus-Click auf eine Komponente werden alle Attribute aufgelistet | | |
| **Abhängigkeit:** | 10 | | |

**Tabelle 13: Anforderung 12 (Attribute anzeigen)**

| **Anforderungs-ID:** | 13 | **Anforderungstyp:** | 10 |
|---|---|---|---|
| **Beschreibung:** | Per Maus-Click auf eine Komponente sollen die Relationen zu anderen Komponenten angezeigt werden. | | |
| **Hintergrund:** | Für eine effektive Nutzung des Metamodell-Browsers muss der User die Möglichkeit haben, sich die Relationen zu anderen Elementen anzeigen zu lassen. | | |
| **Erfüllungskriterium:** | Durch einen Maus-Click auf ein Element wird das Element selbst, die Relationen zu anderen Elementen sowie die anderen Elemente angezeigt. | | |
| **Abhängigkeit:** | 10 | | |

**Tabelle 14: Anforderung 13 (Relationen zu anderen Elementen anzeigen)**

| **Anforderungs-ID:** | 14 | **Anforderungstyp:** | 10 |
|---|---|---|---|
| **Beschreibung:** | Per Maus-Click auf eine Relation in der Elemente-Ansicht sollen die Eigenschaften der Relation angezeigt werden. | | |
| **Hintergrund:** | Für eine effektive Nutzung des Metamodell-Browsers muss der User die Möglichkeit haben, sich die Eigenschaften der gewählten Relationen anzeigen zu lassen. | | |
| **Erfüllungskriterium:** | Nachdem der User auf eine Relation in der Element-Ansicht klickt, wechselt die Applikation in die Relations-Ansicht und zeigt die Eigenschaften der Relation. | | |
| **Abhängigkeit:** | 13 | | |

**Tabelle 15: Anforderung 14 (Relations-Attribute anzeigen)**

| **Anforderungs-ID:** | 15 | **Anforderungstyp:** | 10 |
|---|---|---|---|
| **Beschreibung:** | Per Maus-Click auf eine Relations-Komponente sollen alle Elemente angezeigt werden, zwischen denen diese Relation vorkommt. | | |
| **Hintergrund:** | Für eine effektive Nutzung des Metamodell-Browsers muss der User die Möglichkeit haben, sich die Elemente, zwischen denen die gewählte Relation vorkommt, anzeigen zu lassen. | | |
| **Erfüllungskriterium:** | Nachdem der Anwender auf eine Relations-Komponente klickt, werden alle Elemente von denen die Relation ausgeht, die Relation selber, und alle Elemente wohin die Relation führt, angezeigt. | | |
| **Abhängigkeit:** | 10 | | |

**Tabelle 16: Anforderung 15 (Relation mit Elementen anzeigen)**

| **Anforderungs-ID:** | 16 | **Anforderungstyp:** | 10 |
|---|---|---|---|
| **Beschreibung:** | Per Maus-Click auf ein Element in der Relations-Ansicht sollen dessen Eigenschaften angezeigt werden. | | |
| **Hintergrund:** | Für eine effektive Nutzung des Metamodell-Browsers muss der User die Möglichkeit haben, sich die Eigenschaften des gewählten Elements anzeigen zu lassen. | | |
| **Erfüllungskriterium:** | Nachdem der User auf ein Element in der Relations-Ansicht klickt, wechselt die Applikation in die Elemente-Ansicht und zeigt die Eigenschaften des Elements. | | |
| **Abhängigkeit:** | 15 | | |

**Tabelle 17: Anforderung 16 (Element-Attribute anzeigen)**

| | | | |
|---|---|---|---|
| **Anforderungs-ID:** | 17 | **Anforderungstyp:** | 9 |
| **Beschreibung:** | Per Maus-Click auf ein Attribut soll ein Hilfetext über das jeweilige Attribut angezeigt werden. | | |
| **Hintergrund:** | Für eine effektive Nutzung des Metamodell-Browsers muss der User die Möglichkeit haben, sich den Hilfetext eines gewählten Attributs anzeigen zu lassen. | | |
| **Erfüllungskriterium:** | Nachdem der User auf ein Attribut klickt, wird der Hilfetext in einem Textfeld angezeigt. | | |
| **Abhängigkeit:** | 12 | | |

**Tabelle 18:Anforderung 17 (Hilfetext anzeigen)**

| | | | |
|---|---|---|---|
| **Anforderungs-ID:** | 18 | **Anforderungstyp:** | 9 |
| **Beschreibung:** | Eine Überblickgrafik für das gesamte Metamodell soll angezeigt werden. | | |
| **Hintergrund:** | Um dem User einen Überblick über alle Elemente und deren Relationen untereinander zu verschaffen, soll das gesamte Metamodell als Überblickgrafik angezeigt werden. | | |
| **Erfüllungskriterium:** | Nachdem der User ein Metamodell ausgewählt hat, wird die Überblickgrafik auf einer eigenen Seite angezeigt. | | |
| **Abhängigkeit:** | 9 | | |

**Tabelle 19: Anforderung 18 (Überblicksgrafik anzeigen)**

# 7 Prototyp – Konzept

Dieses Kapitel widmet sich den verschiedenen Entwurfsphasen des Metamodell-Browsers. Zu Beginn stand ein experimenteller Entwurf, in dem erste Designideen versuchsweise umgesetzt wurden und eine erste Rich-Internet-Anwendung auf Basis der Flex-Technologie implementiert wurde. Dieser Entwurf zählt zur **explorativen Phase**. Das anschließende **Grobkonzept** beinhaltet den Entwurf für die Architektur der Applikation sowie die Konkretisierung des Designs der Benutzeroberfläche. Im abschließenden **Feinentwurf** wurden die Interaktionsmöglichkeiten des Benutzers detailliert festgelegt, die benötigten Klassen für die serverseitige Java-Logik bzw. die Komponenten für die Flex-GUI definiert und das Design erstellt.

## 7.1 Explorative Phase

Die explorative Phase diente in erster Linie Textzwecken, um die Darstellungsmöglichkeiten und die Funktionsweise der Flex-Technologie auszuloten. Die Erkenntnisse aus Kapitel 2.2 (Erkenntnistheoretischer Bezugsrahmen) und Kapitel 4 (Adressaten) spielten in dieser Phase noch eine untergeordnete Rolle. Vielmehr wurde mit einer Gesamtdarstellung des Metamodells experimentiert, in dem der Benutzer eine aktive Rolle einnehmen sollte. Aus Gründen der fehlenden wissenschaftlichen Sinnhaftigkeit wurde dieser Ansatz aber wieder verworfen.

### 7.1.1 Funktionsumfang (Experimenteller Entwurf)

Zu Beginn der explorativen Phase standen folgende Überlegungen:

1. Welche Informationen sollen mit dem Metamodel-Browser vermittelt werden?
2. Wie soll die Applikation dargestellt werden?
3. Wie soll die Handhabung des Metamodell-Browsers erfolgen?

Ad 1: Der Metamodell-Browser soll dazu dienen, Metamodelle zu visualisieren. Zu diesem Zweck ist es für einen potentiellen Anwender der Plattform unumgänglich, das zu betrachtende Metamodell auswählen zu können. Nachdem es nicht sehr sinnvoll erscheint, den Anwender mit einem riesigen Pool von Metamodellen zu konfrontieren, sollten Metamodelle aus gleichen Domains in Gruppen zusammengefasst werden. Hierfür würde sich eine Baumstruktur am besten eignen. Hat sich der Anwender für ein Metamodell entschieden, könnten die Komponenten und deren Beziehungen innerhalb eines Boards dargestellt werden.

Um die Übersichtlichkeit zu bewahren und die Komponenten nicht zu groß zu gestalten, werden zu diesem Zeitpunkt jedoch noch nicht die Attribute der Komponenten eingeblendet, sondern erst nach einem Doppel-Klick auf eine Komponente, z. B.: durch ein Pop-Up, würde eine Detailübersicht mit allen Attributen der ausgewählten Komponente dargestellt.

Ad 2: Für die Auswahl des relevanten Metamodells würde sich eine Baumstruktur anbieten. Flex stellt für diese Anforderung eine eigene Tree-Komponente zur Verfügung, wie sie in Abbildung 26 dargestellt ist. Für die Darstellung des Diagramm-Boards, auf dem die Metamodelle präsentiert werden, bietet sich eine einfache BoarderContainer-Komponente an.

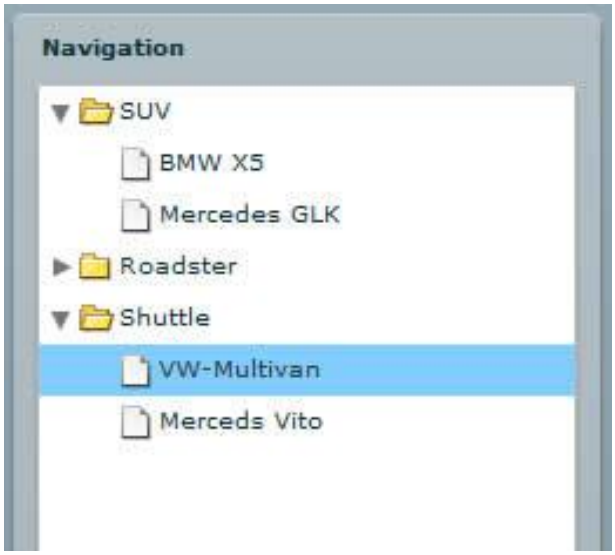


**Abbildung 26: Tree-Komponente in Flex**

Ad 3: Anhand der ersten beiden Punkte könnte man davon ausgehen, dass die Bedienung des Metamodell-Browsers eine sehr einfache ist. Man wählt ein Metamodell aus der Baumstruktur aus, macht einen Maus-Klick darauf, und das Metamodell erscheint mitsamt den Beziehungen auf dem Diagramm-Board. An dieser Stelle wird man bei der Konzeptionierung mit dem ersten großen Problem konfrontiert:

Metamodelle besitzen im Allgemeinen die Eigenschaft, dass sie nicht dafür konzipiert sind, graphisch durch Maschinen dargestellt zu werden. Sie sind eine Sammlung von Komponenten bzw. Klassen und deren Beziehungen, um eine Modellierungssprache zu beschreiben. Das bedeutet in weiterer Folge, dass weder die Komponenten noch die Beziehungen für Maschinen verständliche Informationen beinhalten, die eine graphische Auswertung ermöglichen. Im Klartext bedeutet das: Keines der Elemente hat in seiner Beschreibung x-

bzw. y-Koordinaten hinterlegt, anhand derer eine Positionierung auf dem Graphik-Board erfolgen könnte.

Aus diesem Grund wurde in der explorativen Phase des Metamodell-Browser-Entwurfs die Einbeziehung der gestalterischen  Kreativität der Benutzer in Erwägung gezogen. Dies bedeutet, dass die Benutzer selbst für die Anordnung der Komponenten auf dem Diagramm-Board verantwortlich gewesen wären. Diese Vorgehensweise barg die Notwendigkeit einer zusätzlichen Darstellungs-Komponente in sich, nämlich eines Data-Grids (Abbildung 27).

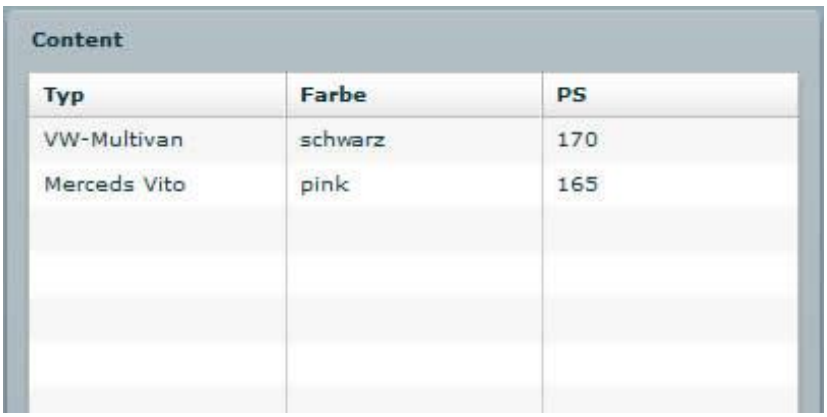


**Abbildung 27: DataGrid-Komponente in Flex**

Der angedachte funktionale Ablauf wäre um den gestalterischen Aspekt des Anwenders erweitert worden. Nachdem der Benutzer der Applikation ein für ihn relevantes Metamodell aus der Baumstruktur ausgewählt hätte, wären die Elemente des Metamodells in einem Data-Grid erschienen. Die hier aufgelisteten Elemente sollte nun der Anwender per Drag & Drop auf dem Graphik-Board nach seinem Gutdünken verteilen können. Die entsprechenden Beziehungen zwischen zwei Klassen wären automatisch eingefügt worden, sobald sich die betroffenen Klassen auf dem Board befänden. Sollten sich Unübersichtlichkeiten durch Überschneidungen ergeben (z. B.: eine Beziehung läuft über eine Komponente), so wäre es dem Anwender frei gestellt gewesen, die Komponenten neu zu arrangieren, indem er sie per Drag & Drop an eine andere Stelle des Graphik-Boards verschiebt. Die Beziehungen sollten an den Komponenten „kleben“ bleiben und automatisch mit den Komponenten mitgezogen werden.

### 7.1.2 Oberflächendesign (Experimenteller Entwurf)

Um eine visuell ansprechende Internet-Applikation zu erstellen, empfiehlt es sich, den ersten Entwurf innerhalb eines Graphikprogramms zu designen. In diesem konnten

Farbkombinationen getestet und aufeinander abgestimmt, Schriften ausgesucht, ein erstes Layout erstellt und mitunter auch Logos entworfen werden. Sollte die Anwendung über mehrere Unterseiten verfügen, könnten diese als so genannte „States“ ebenfalls vorab entworfen werden. Als Entwicklungsumgebung für  den Graphikentwurf eigneten sich Adobe Photoshop oder Fireworks aus dem gleichen Hause.

Als Hintergrund wurde eine graue Fläche verwendet, der mit diversen Filterfunktionen ein metallisches Erscheinungsbild verabreicht wurde. Das Logo bestand aus dem Titel der Applikation und einer Graphik, die einen direkten Bezug zur Open-Model-Initiative herstellen sollte und von der Open-Model-Initiative eigenen Plattform entlehnt wurde. Dabei wurde der Schriftzug ebenfalls mit Filter- und Schattenoptionen bereichert um einen chrom-ähnlichen Effekt zu erzielen (Abbildung 28).

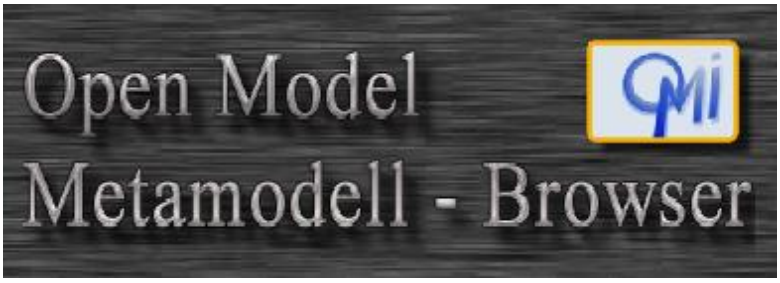


**Abbildung 28: Applikationsbezeichnung mit Logo**

Um die gesamte Applikation farblich interessanter zu gestalten wurde als Kontrast zum grauen Hintergrund für die Rahmen der Komponenten ein sattes Orange verwendet. Ähnliche Orangetöne kamen auch für die Statusdarstellungen (z. B.: Auswahl, Roll-Over)  der Auswahlkomponenten (Tree und DataGrid) zur Anwendung (Abbildung 29).

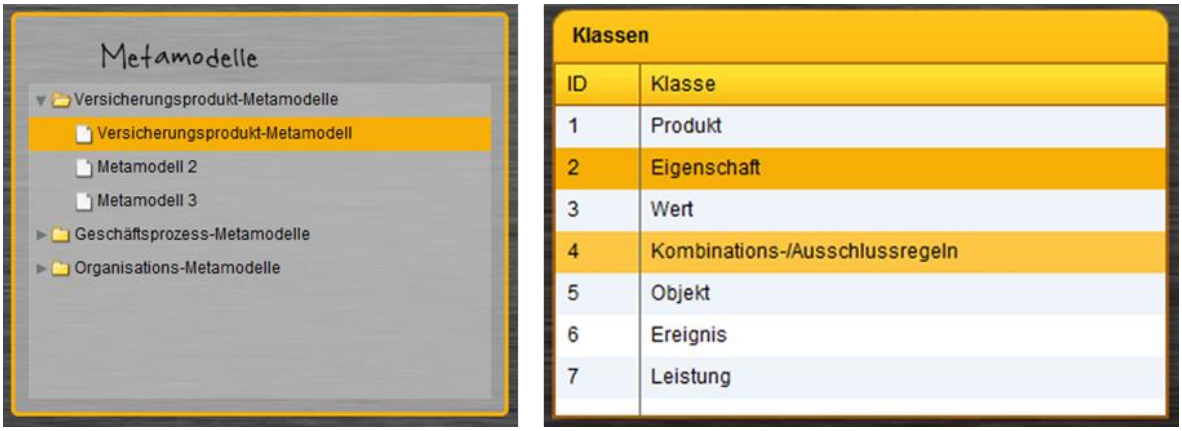


**Abbildung 29: Farbliche Gestaltung der Tree- bzw. der DataGrid-Komponente**

Für die Gestaltung der Metamodelle wurden eigens erstellte Flex-Komponenten verwendet, deren Erscheinungsbild eine Assoziation zu den Klassen in einem UML-Klassendiagramm hervorrufen sollten (Abbildung 30).

**Abbildung 30: Custom-Flex-Komponente für die Darstellung eines Elements**

Insgesamt sollte sich die Darstellung des Metamodells an eine in der Wirtschaftsinformatik gängigen Darstellungsformen orientieren (Abbildung 31).

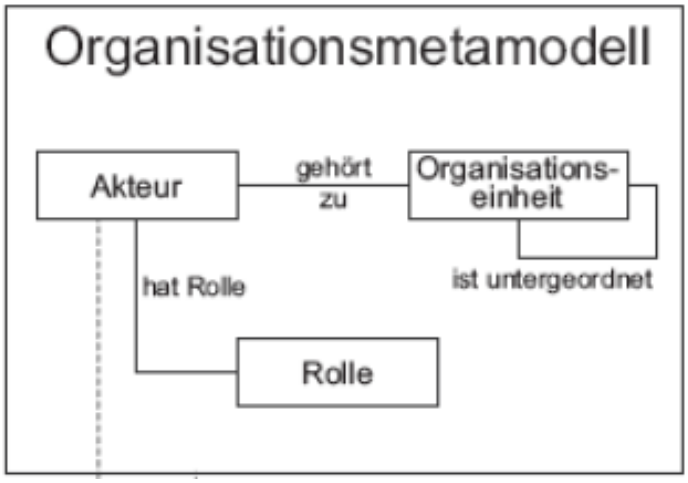


**Abbildung 31: Gängige Darstellung eines Metamodells[21]**

Die folgende Abbildung zeigt eine mögliche Darstellung im Metamodell-Browser des in Abbildung 32 gezeigten Organisationsmetamodells.

[21] http://wit.tuwien.ac.at/teaching/courses/ws05/im_se/metamodels.pdf

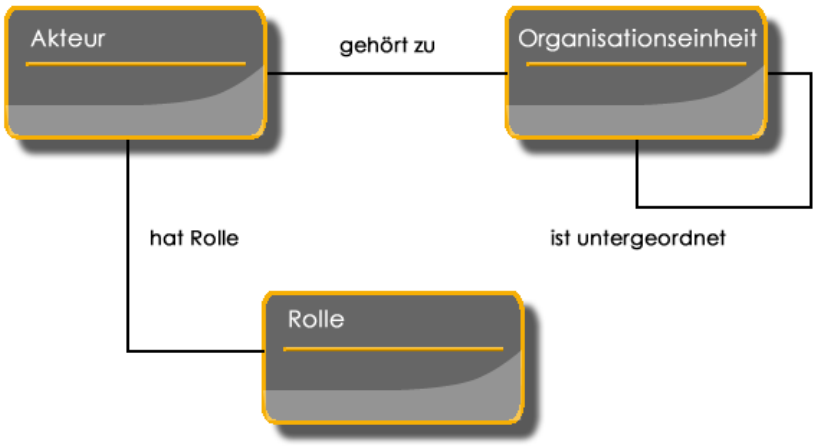


**Abbildung 32: Mögliche Darstellung eines Metamodells im MMB**

Per Doppel-Klick auf eine Klasse sollte es möglich sein, sich die Attribute der Klasse anzeigen zu lassen. Dies sollte mittels eines Pop-Up-Fensters realisiert werden. Sobald der Doppel-Klick ausgeführt würde, würden die Details der Klasse dargestellt (Abbildung 33).

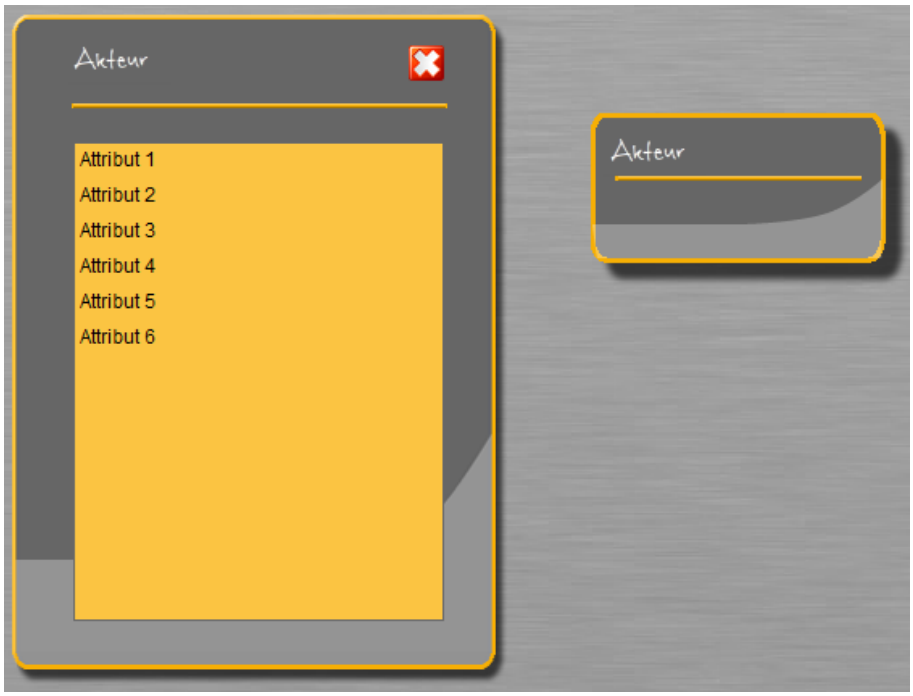


**Abbildung 33: Darstellung des Pop-Ups mit den Elementattributen**

Nachdem sämtliche visuellen Komponenten des Metamodell-Browsers in einem Graphikprogramm designed wurden, konnten diese für den endgültigen Entwurf zusammengestellt werden. Um die Bedienung der Anwendung intuitiv zu gestalten, wurden die Auswahlkomponenten, ähnlich wie bei vielen Web-Seiten die Menüführung, auf der

linken Seite positioniert. Das Logo bekam seinen Platz oberhalb der Auswahlkomponenten in der linken oberen Ecke, ebenfalls in Anlehnung an gängige Website-Layouts. Der gesamte restliche Teil des Bildschirms wurde durch das Graphik-Board eingenommen. Dieses sollte sich nach der Implementierung in der Breite an die Auflösung des Bildschirms anpassen. Abbildung 34 zeigt das fertige Layout des experimentellen Entwurfs.

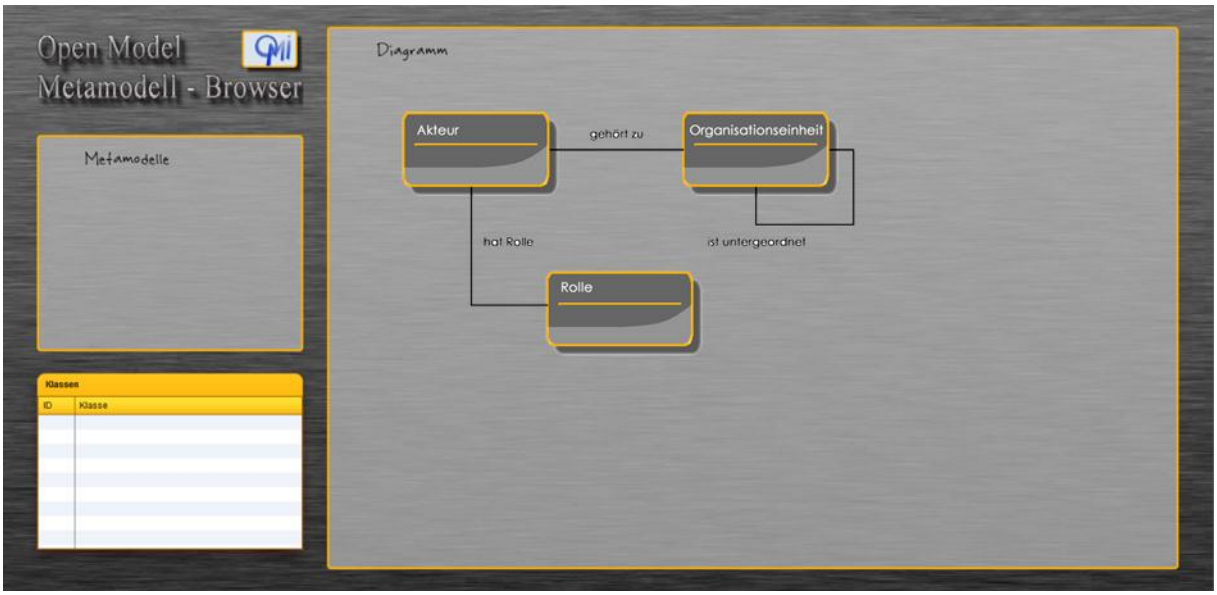


**Abbildung 34: Fertiges Layout des MMB**

### 7.1.3 Überführung des Oberflächendesigns in eine Flex-Anwendung[22] (Experimenteller Entwurf)

Als nächster Schritt musste die in einem Graphikprogramm erstellte Benutzeroberfläche in eine Flex-Anwendung transformiert werden. Prinzipiell würde der von Adobe vorgegebene Weg an dieser Stelle die Weiterentwicklung des Projekts mit Hilfe von Flash-Catalyst vorsehen. Aufgrund der funktionalen Konzeption des Metamodell-Browsers wurden aber in erster Linie Komponenten bei der Oberflächenerstellung verwendet, die von vornherein durch das Flex-Framework angeboten werden. Das bedeutet, dass jede Graphikkomponente des Designs 1:1 durch eine Flex-Komponente ersetzt werden kann, vorausgesetzt, man passt die Styles der Komponente an die zuvor erstellten Designvorgaben an, und somit der Umweg über Flash-Catalyst entfällt.

[22] Auf eine Beschreibung, wie ein Flex-Projekt angelegt wird und wie ein Tomcat-Server mit BlazeDS-Technologie in das Projekt integriert wird, wird im Zuge der vorliegenden Arbeit und mit dem Hinweis auf unzählige Tutorials im WWW verzichtet.

Nichts desto trotz soll an diesem Punkt kurz auf dieses neue Entwicklungstool eingegangen werden.

**Exkurs Flash Catalyst**

Mit der Veröffentlichung der Creative-Suite 5 stellt Adobe ein Werkzeug zur Verfügung, das den Bruch zwischen einer Photoshop- bzw. Fireworksvorlage und der echten Applikation verhindern soll. In vielen Fällen folgt die Entwicklung einer Internet-Anwendung folgender Prozedur:

1. Die Anforderungen an die Anwendung werden in Zusammenarbeit mit einem Kunden definiert.
2. Durch einen Designer wird eine Oberfläche erstellt, welche die Anforderungen graphisch erfüllen kann. Es ist durchaus denkbar, dass dieser Prozess von einer externen Graphikagentur durchgeführt wird.
3. Danach kann die Oberfläche als Prototyp dem Kunden vorgelegt werden. Mit Hilfe dieses Prototyps kann sich der Kunde ein Bild über das Aussehen, über die Navigationslogik, etc. machen.
4. Sobald der Kunde mit dem Ergebnis einverstanden ist, kann die Entwicklerfirma damit beginnen, die Oberfläche browserkonform umzusetzen.

Dieses „Nachbauen“ einer Photoshopvorlage weicht in den meisten Fällen aber erheblich vom ursprünglichen Design des Entwurfs ab. Mit einer reinen HTML/JavaScript/AJAX-Applikation ist die Wahrscheinlichkeit, dass die fertige Anwendung dem Prototyp zumindest entfernt ähnlich sieht, noch einigermaßen gegeben. Ein exaktes Abbild zu erzeugen, ist aber äußerst unwahrscheinlich.

Diese Problematik soll durch Flash-Catalyst gelöst werden. Mit Flash-Catalyst ist es möglich, den Graphiken einer zuvor geladenen Photoshop-Datei die entsprechenden Funktionen in der Web-Anwendung zuzuweisen. So kann zum Beispiel das Aussehen von Scroll-Balken oder Listenelementen völlig individuell gestaltet werden (Abbildung 35), um im Anschluss daran auf verhältnismäßig einfache Weise deren Funktion zu definieren.

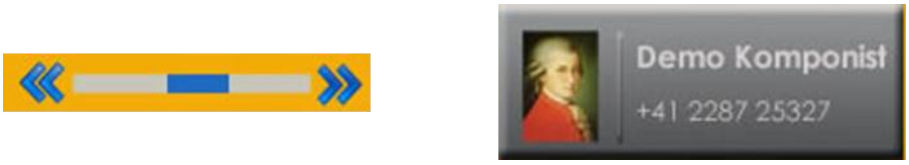


**Abbildung 35: Beispiele für die Gestaltung eines Scroll-Balkens bzw. eines Listenelements**

Um das Beispiel des Scroll-Balkens noch einmal aufzugreifen; dieser muss aus folgenden vier Komponenten bestehen, denen die einzelnen Funktionen zugewiesen werden:


….....Scroll-Button nach links


….....Scroll-Button nach rechts


….....Scroll-Schiene


….....Scroll-Schieberegler

Darüber hinaus können in Flash-Catalyst einzelnen Graphikelementen diverse Effekte zugewiesen werden, z. B.: Glow-Effekte für Buttons oder Fading-Effekte beim Wechsel zwischen verschiedenen States der Anwendung.

Sobald sämtliche graphikspezifischen Funktionen zugeordnet wurden, kann das Catalyst-Projekt als .fxp-Datei gespeichert werden. Die Dateiendung weist schon darauf hin, dass das Ausgabeformat des Catalyst-Projekts im Endeffekt ein Flex-Projekt ist und somit direkt in den Flash-Builder zur Weiterverarbeitung importiert werden kann. Im Flash-Builder kann in weiterer Folge die gesamte Programm- bzw. Serverlogik implementiert und mit den Bedienelementen der graphischen Oberfläche verbunden werden.

Diese von Adobe angedachte Entwicklungskette für eine Flex-Anwendung erleichtert das Transformieren eines ersten grafischen Prototyps in eine fertige Web-Anwendung erheblich. Zusätzlich wird es möglich, diese Transformation ohne Einbußen bzw. Kompromisse das Design betreffend, durchzuführen. So wie die Oberfläche vom Designer entworfen wurde, wird sie auch später auf dem Bildschirm dargestellt. Diese Art der Entwicklung einer Internet-Anwendung kann als sehr innovativ bezeichnet werden. Sobald man eine erste Applikation auf diesem Weg erstellt hat, wird man jedoch sehr schnell mit dem vorhandenen Gefahrenpotential konfrontiert.

Die Entwicklung einer Flex-Anwendung auf  dem Weg Photoshop-Catalyst-FlashBuilder folgt dem klassischen Wasserfall-Prinzip (Abbildung 36), und dieses ist (nicht zu Unrecht) nicht der Weisheit letzter Schluss und somit in Entwicklerkreisen auch nicht gern gesehen. Die Problematik besteht in dem Rattenschwanz von Nachbesserungen, den jede Änderung an der Applikation nach sich zieht. Befindet man sich beispielsweise schon mitten in der Implementierung der Applikationslogik und man bemerkt, dass man vergessen hat, in Catalyst für eine Graphikkomponente deren Funktion in der Flex-Anwendung zu definieren, so ist es

nicht möglich, dieses Versäumnis in Catalyst einfach nachzuholen. Dies resultiert daraus, dass die Flex-Applikationslogik in das fertige Catalyst-Projekt integriert wird, d. h. sobald das überarbeitete Catalyst-Projekt neu importiert wird, sind alle Änderungen und Anpassungen, die bereits durchgeführt wurden, hinfällig.

Ein noch schlimmeres Szenario entsteht, wenn kurz vor der Fertigstellung der Applikation festgestellt wird, dass eine graphische Komponente in der Photoshop-Designvorlage vergessen wurde bzw. nachträglich integriert werden soll. In diesem Fall müsste die gesamte Entwicklungskette noch einmal von vorne durchlaufen werden, sollte man nicht in der Lage sein, die Anpassungen direkt im Source-Code der Flex-Anwendung durchzuführen.

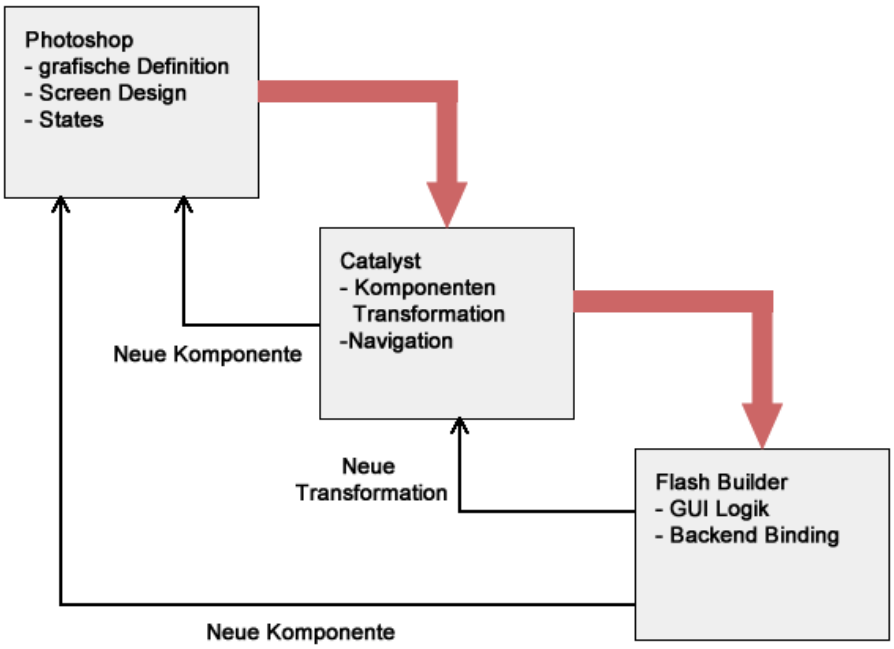


**Abbildung 36: Wasserfallmodell einer Flex-Entwicklungskette [vgl. MÜLLER2010]**

**Fortsetzung Kapitel 9.1.3**

Wie bereits erwähnt, wurde für die Realisierung der Benutzeroberfläche des experimentellen Entwurfs die Verwendung von Catalyst nicht benötigt, da es sich bei den verwendeten Komponenten ausschließlich um Standardkomponenten des Flex-Frameworks handelt. Die Gestaltung der Oberfläche konnte entweder direkt in MXML durchgeführt werden (durch die XML-Syntax ist dies verhältnismäßig leicht umzusetzen), oder man bediente sich des Design-

Editors, in dem die Komponenten nach dem WYSIWYG-Prinzip[23] angeordnet werden können (Abbildung 37).

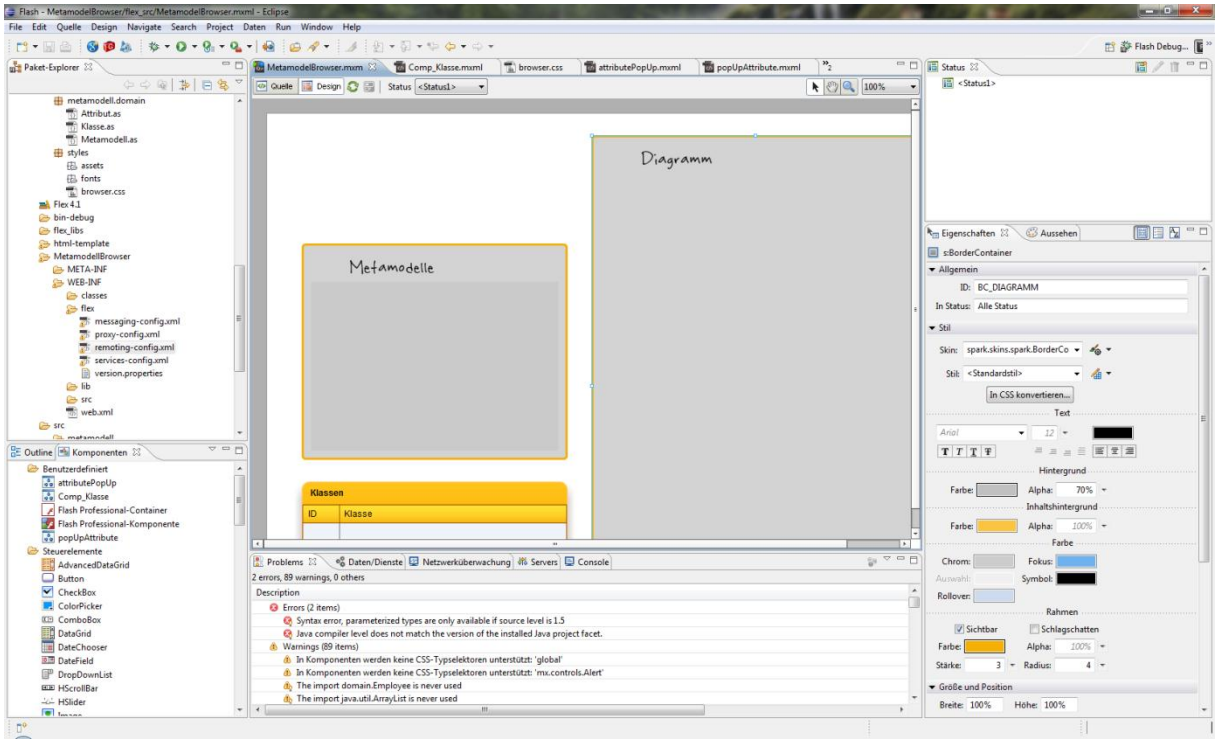


**Abbildung 37: Screenshot – Komponentenanpassung im Design-Editor**

Auch wenn der Design-Editor für die Positionierung der Komponenten nicht unbedingt erforderlich ist, so ist er doch eine große Hilfe bei der Vergabe der Style-Attribute, mit denen die Farben, der Rahmen, etc. der Komponenten individuell gestaltet werden können.

Für erste Tests lieferte eine in ActionScript implementierte Routine Dummy-Daten an den Metamodel-Browser.

Dieser experimentelle Entwurf wurde im Anschluss mit Mitgliedern der Open-Model-Community mit dem Ergebnis diskutiert, dass dieser erste Entwurf weder in der Darstellung noch in der Anwendung für wissenschaftliches Arbeiten geeignet war. Der Metamodell-Browser sollte als wichtigstes Merkmal einen schnellen Überblick über ein Metamodell bieten. Durch das Zusammenfügen der Elemente durch den Benutzer war diese Anforderung aber in der momentanen Entwicklungsphase nicht erfüllt. Trotzdem konnten viele Erkenntnisse, die bei der Erstellung des experimentellen Entwurfs gewonnen wurden, für die

[23] WYSIWYG: steht für **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et und bedeutet: so wie es im graphischen Editor angezeigt wird, so wird es auch später im Browser dargestellt.

weitere Arbeit verwendet werden. Insbesondere wurde bereits eine mögliche Darstellungsform für einen zukünftigen, selbstgenerierten Gesamtüberblick über das Metamodell entwickelt.

## 7.2 Grobkonzept

Die Basis des Metamodel-Browsers bildet eine 3-Schichten-Architektur. Die drei Schichten werden aus dem Flex-basierten User-Interface (Präsentationsschicht), der serverseitigen Java-Logik (Logikschicht) und der Datenhaltungsschicht gebildet. Die Datenschicht wird im Falle des Prototyps durch zwei Verzeichnisse repräsentiert, wobei ein Verzeichnis die XML-Files mit den Daten der Metamodelle beinhaltet und das zweite Verzeichnis die dazugehörenden Graphiken zur Verfügung stellt (Abbildung 38).

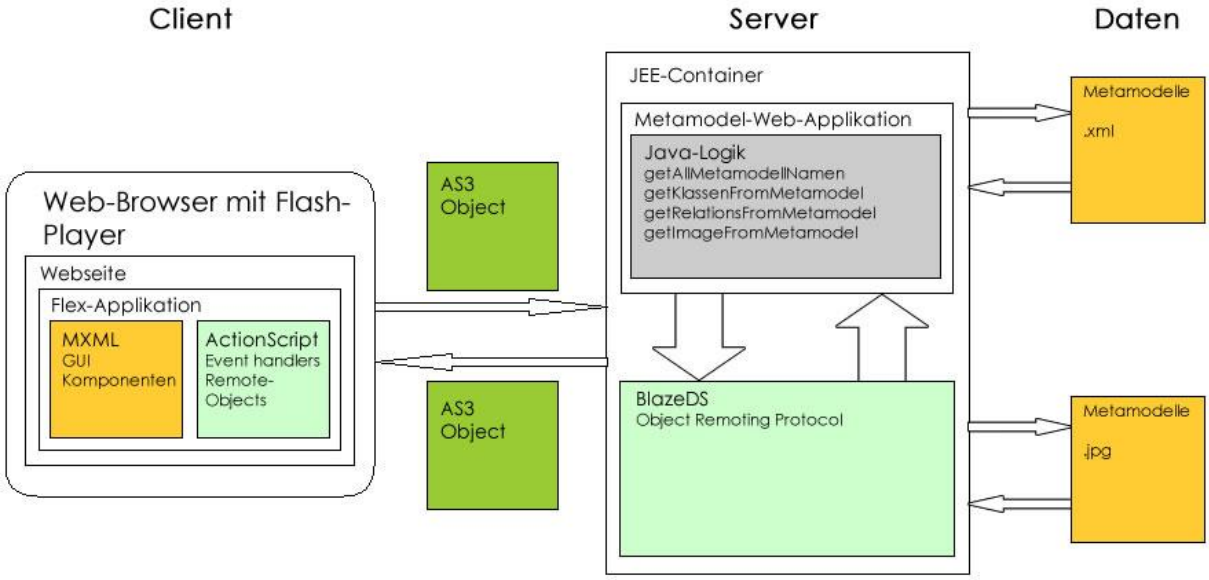


**Abbildung 38: Metamodel-Browser DataFlow-Diagramm**

Die Kommunikation zwischen Client und Server geschieht über die J2EE-Erweiterung BlazeDS. Bei BlazeDS handelt es sich um ein Servlet, welches den Datentransfer zwischen Flex-Applikation und Server steuert. Ein beträchtlicher Vorteil von BlazeDS liegt darin, dass Java-Objekte direkt in ActionScript-Objekte umgewandelt werden können. Des Weiteren werden mehrere Java-Grunddatentypen direkt unterstützt und können somit ohne den Umweg der Serialisierung bzw. Deserialisierung, wie es zum Beispiel bei der Übertragung durch

einen XML-Stream notwendig wäre, erfolgen. Dadurch ist eine erhebliche Performance-Verbesserung erreichbar. Folgende Liste zeigt einige der unterstützten Datentypen[24]:

| **Java-Datentyp** | **ActionScript-Datentyp** |
|---|---|
| java.util.List | Array |
| java.util.Map | Array |
| boolean, java.lang.Boolean | Boolean |
| byte[] | Flash.utils.ByteArray |
| integer, java.lang.Integer | int |
| java.lang.String | String |
| typisiertes Objekt (z. B. Relation.java) | typisiertes Objekt (z.B. Relation.as) |
| org.w3c.dom.Document | XML |

**Tabelle 20: Auszug unterstützter Datentypen**

Bei der Implementierung des Metamodell-Browsers werden in erster Linie Arrays zur Anwendung kommen, welche die Objekte der Klassen und Relationen beinhalten. Für die Übergabe der Übersichtsgrafik wird ein ByteArray Verwendung finden.

Um das XML-File eines Metamodells auszulesen, wird in der Java-Logik ein DOM-Parser implementiert, der den gesamten XML-Baum in den Speicher liest. Dadurch sind schnelle Zugriffe auf die einzelnen Elemente und deren Überführung in Java-Objekte garantiert. Die Wahrscheinlichkeit, dass ein Metamodell im XML-Format zu groß für den Speicher ist, sodass der DOM-Parser an seine Grenzen stoßen würde, ist äußerst gering.

Die Metamodell-Übersichtsdatei im jpg-Format wird durch eine Java-Methode als FileInputStream eingelesen und in ein Java-ByteArray geschrieben. Dadurch kann es, wie aus

[24] Vgl. Florian Müller S.122

Tabelle 20 ersichtlich, 1:1 in ein ActionScript-ByteArray transformiert werden und so im Flex-Frontend angezeigt werden.

## 7.3 Feinkonzept

Der Feinentwurf resultiert sowohl aus den Analysen in Kapitel 4 (Erstellung eines erkenntnistheoretischen Bezugsrahmens) und Kapitel 6 (Die Open Model Community als vorrangige Adressaten) sowie den Gesprächen mit Chef- und Co-Entwicklern der Open-Model-Community und der Aufarbeitung der Designschwächen des experimentellen Entwurfs.

### 7.3.1 Feinkonzept der Flex-Benutzeroberfläche

Anhand der gewonnenen Erkenntnisse und der Anforderungsanalyse wurde die Benutzeroberfläche neu gestaltet. Eine Kombination aus gelisteten Elementgraphiken sowie ein Abbild des gesamten Metamodells sollen es dem Benutzer ermöglichen, die relevanten Informationen über die einzelnen Elemente sowie über deren Beziehungen durch den Metamodel-Browser zu erhalten. Zusätzlich sollen die Attribute der Elemente gelistet und, wenn vorhanden, Informationstexte über die einzelnen Attribute angezeigt werden (Abbildung 39).

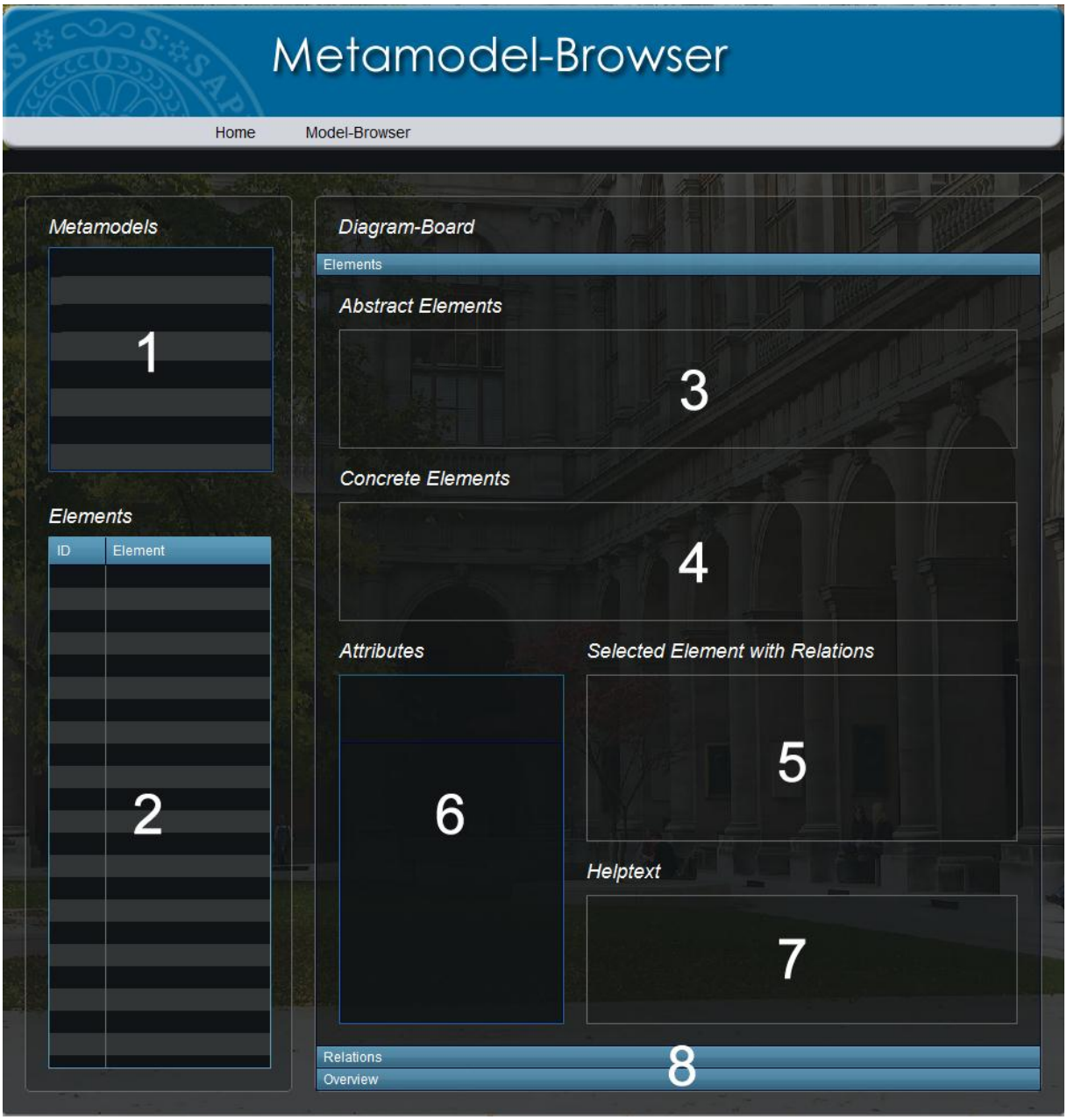


**Abbildung 39: Feinentwurf der Benutzeroberfläche**

Add 1: Wie schon im experimentellen Entwurf soll eine Tree-Komponente dazu dienen, die zur Verfügung stehenden Metamodelle aufzulisten. Mit dieser Komponente ist es dem Benutzer möglich, das gewünschte Metamodell auszuwählen.

Add 2: Ebenfalls wie im experimentellen Entwurf sollen die Elemente des Metamodells in einer DataGrid-Komponente aufgelistet werden. Im Gegensatz zum experimentellen Entwurf dient das DataGrid im Feinentwurf

ausschließlich zur Navigation. Mit einem Mausklick auf ein Element soll dieses in der Grafikliste 3 bzw. Grafikliste 4 angezeigt werden.

Add 3 + 4: In diesen beiden Containern sollen die Elemente des Metamodells grafisch dargestellt werden. Um eine bessere Übersicht zu gewährleisten, werden die Elemente in abstrakte und konkrete Elemente unterteilt.

Add 5: Sobald ein Element ausgewählt wurde, sollen in diesem Container das Element selbst sowie alle Elemente, mit denen dieses Element durch eine Relation verbunden ist, angezeigt werden. Zusätzlich sollen auch die Relationen grafisch dargestellt werden.

Add 6: Einhergehend mit der Auswahl eines Elements sollen in dieser Listen-Komponente sämtliche Attribute des gewählten Elements angezeigt werden.

Add 7: Nach einem Klick auf ein Attribut soll in diesem Textfeld der Beschreibungstext des Attributs erscheinen

Add 8: Dabei handelt es sich um eine Accordion-Komponente. Diese ermöglicht den Wechsel zur Relationsansicht, die ähnlich der Elementansicht gestaltet sein wird, bzw. zum Overview, in dem das gesamte Metamodell als Bitmap dargestellt wird.

Die grafische Notation für die abstrakten und konkreten Elemente bzw. für die Relationen wird wie folgt angezeigt werden (Abbildung 40):

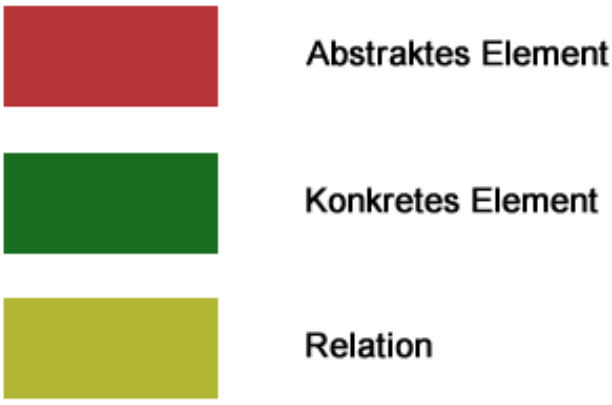


**Abbildung 40: Notation für Elemente und Relationen**

### 7.3.2 Feinkonzept der serverseitigen Java-Logik

Um sämtliche relevante Daten aus dem XML-File auszulesen und in weiterer Folge dem Flex-Frontend zur Darstellung zu übergeben, werden für die Implementierung ein Interface, eine Serviceklasse und vier Objektklassen benötigt (Abbildung 41).

Die folgenden vier Methoden der Serviceklasse bereiten die Daten aus dem XML-File für die weitere Verarbeitung auf:

1. getAllMetamodelNames(): Diese Methode wird initial mit Start des Metamodell-Browsers aufgerufen und liest die Dateinamen der Metamodell-XML-Files aus dem Metamodell-Verzeichnis aus. Diese werden als ArrayList an die Flex-Applikation zurückgegeben.
2. getElementsFromMetamodel(): Wenn der Benutzer ein Metamodell ausgewählt hat, werden mit dieser Methode die Elemente des Metamodells ausgelesen. Der Übergabewert ist der Name des Metamodells. Der Rückgabewert ist eine ArrayList mit den Objekten der Elemente.
3. getRelationsFromMetamodel(): Unmittelbar nach dem Einlesen der Elemente wird diese Methode aufgerufen, um die Relationen aus dem XML-File auszulesen. Übergabewert ist ebenfalls der Name des gewählten Metamodells. Der Rückgabewert an die Flex-Applikation ist eine ArrayList mit den Objekten der Relationen.
4. getImageFromMetamodel(): Um das gesamte Metamodell grafisch darzustellen, liest diese Methode eine Bitmap des Metamodells aus dem Grafikverzeichnis aus. Übergabewert ist wiederum der Name des Metamodells, Rückgabewert ist ein ByteArray.

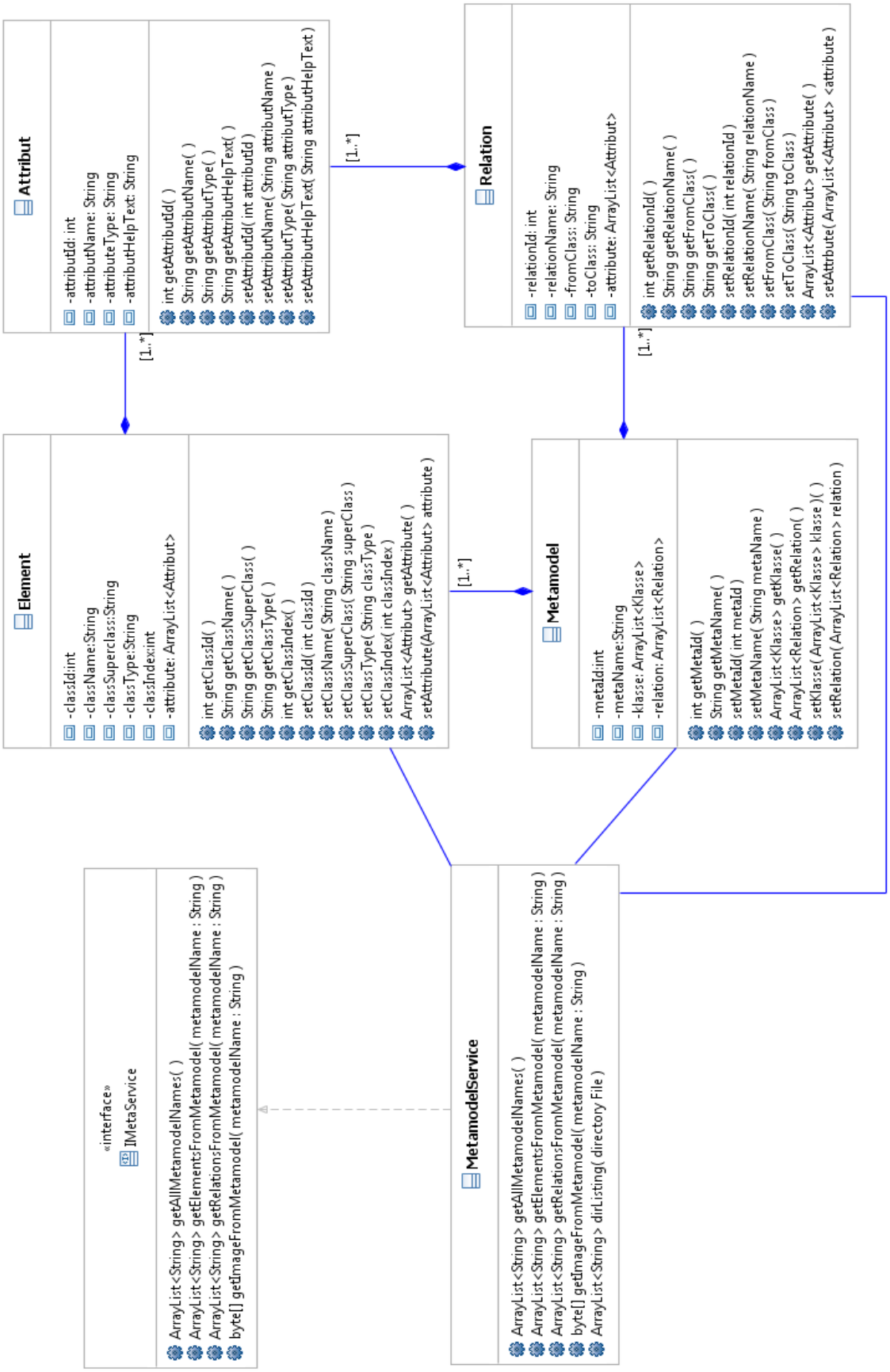


**Abbildung 41: Metamodel-Browser-Service Klassendiagramm**

# 8 Metamodel-Browser: Implementierung

Nachdem die Designschwächen des experimentellen Entwurfs überarbeitet und dem Anforderungskatalog entsprechend angepasst wurden, widmet sich dieses Kapitel der Implementierung des Metamodel-Browsers. Beginnend mit der Programmierung der Java-Logik wird im Anschluss die Flex-GUI entwickelt. Danach werden die benötigten Flex-Komponenten erstellt und die erforderlichen Remote-Zugriffe für die Datenbereitstellung in ActionScript implementiert. Ebenfalls in ActionScript werden die Methoden und Effekte implementiert, die dem Benutzer die geforderten Informationen in einer visuell ansprechenden Weise zur Verfügung stellen.

## 8.1 Implementierung der serverseitigen Java-Logik

### 8.1.1 Objekt-Klassen

Um aus den Daten des XML-Files Java-Objekte für die weitere Verarbeitung zu instanziieren, müssen zunächst die erforderlichen Klassen erstellt werden. Für die Implementierung des Prototyps sind folgende vier Klassen erforderlich:

- Metamodell: Diese Klasse beinhaltet eine Metamodell-ID (metaId) als numerischen Wert (Integer), einen Metamodell-Namen (metaName) als Zeichenfolge(String) und eine Liste(ArrayList) von Elementobjekten (element).
- Element: Für die Bereitstellung der Elementobjekte (element) wird eine Java-Klasse mit der Element-ID (klasseId) als Integer, dem Element-Namen (elementName) als String und eine Liste von Attributobjekten(attribute) benötigt.
- Relation: Relationen beinhalten eine Relations-ID (relationId) als Integer und einen Namen (relationName) als String. Um zu ermitteln, welche Elemente durch die Relation verbunden sind, werden diese beiden Elemente (fromElement, toElement) als Strings angegeben. Die Attribute-Objekte werden als ArrayList übernommen.
- Attribut: Die Attribut-Klasse verfügt über eine Attribut-Id (AttributId) als Integer und einen Attribut-Namen (AttributName) als String.

Wie man anhand dieser Auflistung erkennen kann, liegt hier eine Verschachtelung der Klassen vor. Diese Verschachtelung resultiert daraus, dass jedes Metamodell über zumindest ein oder mehrere Element (e) verfügt und jedes Element zumindest ein oder auch mehrere Attribute beinhalten kann.

Im Anhang ist exemplarisch der Quellcode für die Klasse Relation aufgelistet (Listing 7).

### 8.1.2 Service-Klasse

Im nächsten Schritt wird die Service-Klasse mit den im Feinentwurf definierten Methoden implementiert.

- *getAllMetamodellNames:* In dieser Methode wird zunächst der Pfad zum Verzeichnis festgelegt, in welchem sich die XML-Files der Metamodelle auf dem Tomcat-Server befinden. Sollte das Verzeichnis nicht existieren, wird eine entsprechende Fehlermeldung ausgegeben. Über eine zusätzliche Methode (dirListing), deren Übergabeparameter der Pfad ist, werden alle File-Namen des angegebenen Verzeichnisses in die ArrayList<String> metamodellNames geschrieben. Diese ArrayList ist zugleich der Rückgabewert an das Flex-Frontend (Anhang Listing 8).
- *getElementsFromMetamodel - getRelationsFromMetamodel*[25]*:* mit Hilfe eines DOM-Parsers wird das XML-File eines Metamodells ausgelesen. Unter Verwendung von for-Schleifen werden Element- bzw. Relationsobjekte und die dazugehörenden Attribut-Objekte instanziiert und in weiterer Folge werden die entsprechenden Werte in die Objekte geschrieben. Das folgende Listing veranschaulicht anhand der Kommentare exemplarisch die Funktion der *getRelationFromMetamodel()*-Methode (Anhang Listing 9).
- *getImageFromMetamodel:* Diese Methode dient dazu, eine Gesamtübersicht des entsprechenden Metamodells im Flex-Frontend darzustellen. Diese Abbildungen sind im JPEG-Format im Verzeichnis *Metamodel_Overview* am Tomcat-Server hinterlegt. Die Bezeichnung der Datei muss dabei folgender Konvention folgen:
  Bezeichnung der Metamodell-Datei (ohne die .xml-Endung) plus die Erweiterung *_overview.jpg*.
  Beispiel: heißt die Metamodell-Datei *iSTARMethod_v1.05.xml*, so muss die dazugehörende Bitmap-Datei als *iSTARMethod_v1.05_overview.jpg* bezeichnet werden.
  Sofern die Methode in der angegebenen Datei eine entsprechende Bitmap-Grafik findet, wird diese über einen FileInputStream in einem ByteArray gespeichert und an

[25] Diese beiden Methoden sind sich in der Funktion sehr ähnlich, weshalb auf ein Listing der getElementsFromMetamodel()-Methode verzichtet wird.

das Flex-Frontend als Rückgabewert übergeben. Wird die Datei nicht gefunden, wird eine entsprechende Fehlermeldung ausgegeben (Anhang Listing 10).

### 8.1.3 Interface

Für den Prototyp des Metamodell-Browsers ist die Implementierung eines Interfaces nicht zwingend erforderlich. Um zukünftigen Erweiterungen den einfachen Zugriff auf die Methoden der Java-Service-Klasse zu ermöglichen, wurde es vorsorglich im Prototyp integriert (Anhang Listing 11). Somit können zukünftige Applikationen auf die Funktionen der Service-Klasse zugreifen.

Der Grund, warum das Interface bei der Entwicklung des Prototyps nicht unbedingt erforderlich ist, ist jener, dass bei BlazeDS die Kopplung zwischen Flex-Frontend und Java-Backend nicht über das Interface, sondern direkt über die Java-Klassen erfolgt [MÜLLER2010]. Deshalb muss die Java-Service-Klasse in der *remoting-config.xml* des Flex-Frontends gesondert deklariert werden (Listing 2).

```
<?xml version="1.0" encoding="UTF-8"?>
<service id="remoting-service"
    class="flex.messaging.services.RemotingService">

    <adapters>
        <adapter-definition id="java-object"
      class="flex.messaging.services.remoting.adapters.JavaAdapter"
      default="true"/>
    </adapters>

    <default-channels>
        <channel ref="my-amf"/>
    </default-channels>

    <destination id="MetamodellService">
      <properties>
            <source>metamodell.services.MetamodellService</source>
            <scope>session</scope>
      </properties>
    </destination>
</service>
```

**Listing 2: remoting-config.xml**

## 8.2 Flex-Frontend Implementierung

Die Implementierung des Flex-Frontends setzt sich aus mehreren Teilprojekten zusammen. Zu Beginn steht die Erstellung der Benutzeroberfläche in MXML nach dem Vorbild des Design-Feinentwurfs. Darauf aufbauend werden die grafischen Flex-Komponenten für die abstrakten und konkreten Elemente sowie der Relationen entwickelt. Um die Daten, die in Form von Java-Objekten vom Server abgerufen werden, 1:1 weiterverwenden zu können, werden die Java-Objektklassen in Form von ActionScript-Klassen nachgebildet. Die in der Anforderungsanalyse gewünschte Funktionalität wird im Anschluss durch ActionScript-Methoden implementiert. Der ActionScript-Code entspricht somit der GUI-Logik.

### 8.2.1 MXML-Benutzeroberfläche

Die Erstellung einer Benutzeroberfläche mit MXML gestaltet sich ähnlich wie der Aufbau einer Website mit HTML-Tags. Das endgültige Erscheinungsbild der Applikation resultiert aus der Verschachtelung der verschiedenen Flex-Komponenten. Zusätzlich können bei jeder Flex-Komponente eine Vielzahl an Attributen definiert werden, die direkten Einfluss auf das Aussehen der Komponente haben. Diese Attribute entsprechen teilweise den CSS-Definitionen einer HTML-Datei. Folgendes Beispiel der Tree-Komponente (Abbildung 42), welche die Metamodelle anzeigt, soll die zugrundeliegende Philosophie von MXML veranschaulichen:

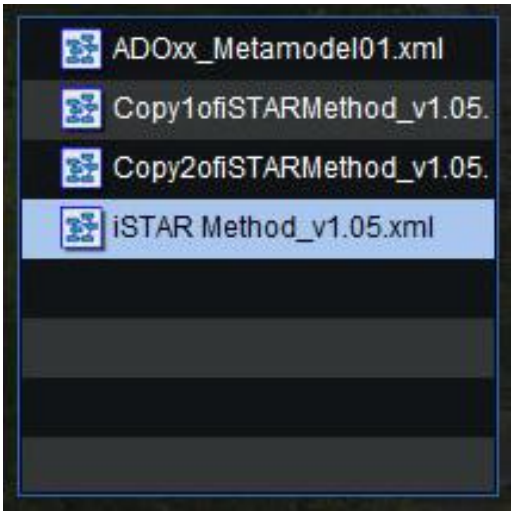


**Abbildung 42: Graphische Darstellung einer Tree-Komponente**

Und hier das Listing der MXML-Definition:

```
<mx:Tree
id="TREE_Metamodelle"
width="200"
height="200"
contentBackgroundColor="0x111416"
borderVisible="true"
borderColor="#2d6988"
click="onSelectMetamodel()"
rollOverColor="0x65a2c1"
chromeColor="#2d6988"
color="#FFFFFF"
alternatingItemColors="[#111416, #323535]"
defaultLeafIcon="@Embed(source='styles/assets/MetamodellIcon_klein.png')"
/>
```

**Listing 3: MXML - Tree Definition**

Anhand des Listings wird die Funktionsweise ersichtlich. Zu Beginn steht die Bezeichnung der Komponente; in diesem Fall *mx:Tree*. Das Attribut *id* macht die Komponente eindeutig identifizierbar. Dies ist besonders wichtig, um die Komponenten mit Daten zu füllen. Einer Tree-Komponente kann man als Datenquelle z. B.: eine ArrayCollection zuordnen. Die weiteren Attribute sind fast ausschließlich für das Aussehen der Komponente zuständig und zum Großteil selbsterklärend. Eine Ausnahme bildet das *click*-Attribut. Das *click*-Attribut teilt der Komponente mit, was zu tun ist, wenn ein einfacher Maus-Klick auf die Komponente erfolgt. In diesem Fall wird die Methode *onSelectMetamodel()* aufgerufen.

Bei zunehmender Verschachtelung wird es immer schwieriger, die Übersicht zu behalten, wie am Beispiel der NavigatorContent-Komponente (Listing 4), welche für die Darstellung der Übersichtsgrafik zuständig ist, ersichtlich wird:

```
<s:NavigatorContent label="Overview" width="100%" height="100%"
id="AKKORD_Overview" backgroundAlpha="0.3">
      <s:BorderContainer left="20" top="20" right="20" bottom="20"
width="100%" height="100%" backgroundAlpha="0.0">
            <s:Scroller id="SCROLL_Overview" width="100%" height="100%">
                  <s:Group>
                        <s:BorderContainer id="BC_Overview" minWidth="1"
```

```
minHeight="1" borderVisible="false" backgroundAlpha="0.0">
                                <mx:Image x="0" y="0"
id="imageOverviewBoard"/>
                        </s:BorderContainer>
                </s:Group>
            </s:Scroller>
      </s:BorderContainer>
</s:NavigatorContent>
```

**Listing 4: Verschachtelte NavigatorContent-Komponente**

Nach dem Erstellen der gesamten Benutzeroberfläche wurde der Applikation ein zusätzlicher *State* hinzugefügt. *States* entsprechen bei Flex-Applikationen den verschiedenen Seiten einer HTML-Website. Der Wechsel zwischen verschiedenen *States* kann mit Hilfe von *LinkButtons* bewerkstelligt werden (Listing 5). Der zusätzliche State sollte im Falle des Metamodel-Browsers eine Begrüßung sowie eine Bedienungsanleitung beinhalten (Abbildung 43).

```
<!-- States definieren -->
<s:states>
      <s:State name="st_home"/>
      <s:State name="st_applikation"/>
</s:states>


<!-- Menueeinträge erstellen -->
      <s:BorderContainer x="146" y="100" width="100%" height="25"
backgroundAlpha="0.0" borderVisible="false">
            <mx:LinkButton x="24" y="3" label="Home" width="80"
click="currentState='st_home'" color="#000000" rollOverColor="#65a2c1"
selectionColor="#006699" fontSize="14"/>
            <mx:LinkButton x="114" y="3" label="Model-Browser" width="120"
click="onChangeApplicationState()" color="#000000" rollOverColor="#65a2c1"
selectionColor="#006699" fontSize="14"/>
      </s:BorderContainer>
```

**Listing 5: Definition der States und der LinkButtons**

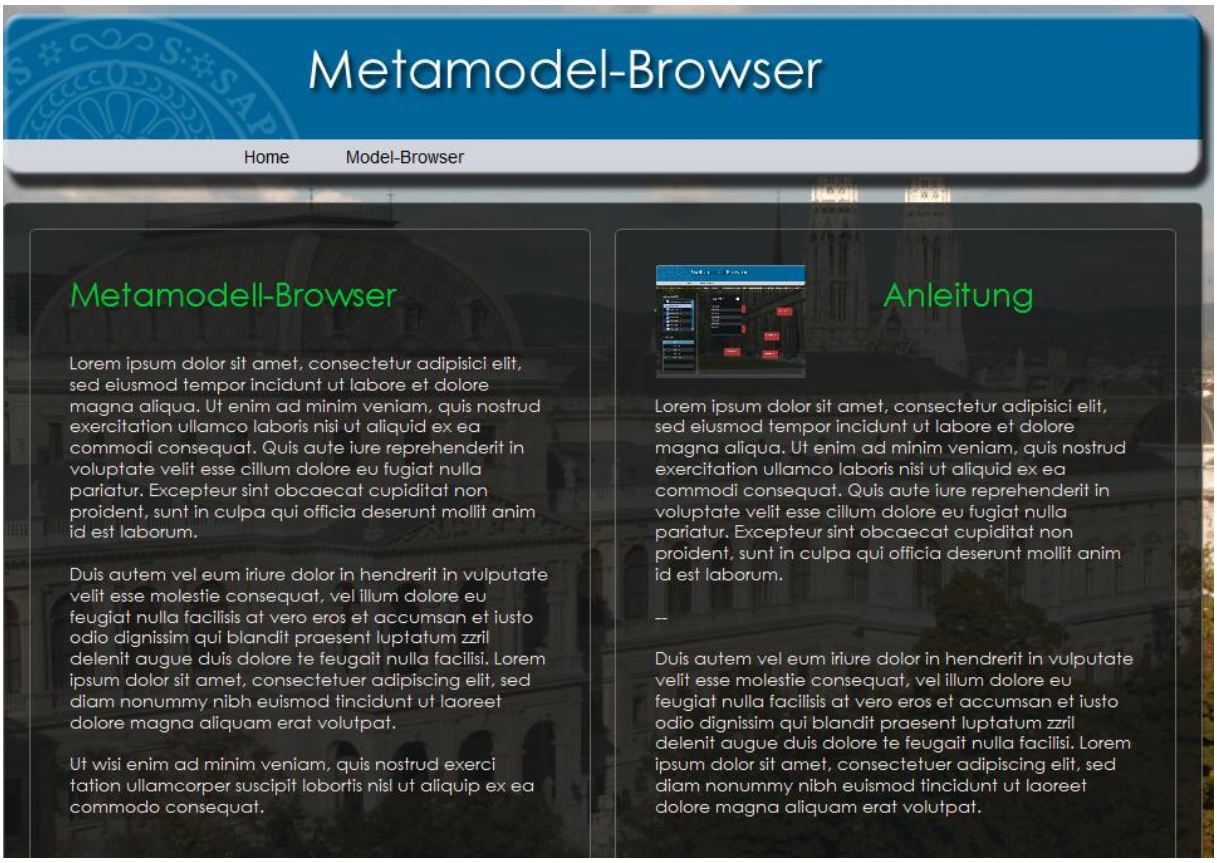


**Abbildung 43: Start-State mit Begrüßung und Anleitung (noch gefüllt mit Blindtext)**

Nachdem die Benutzeroberfläche fertig zusammengestellt war, wurden noch Effekte für ein visuell ansprechenderes Auftreten definiert. Effekte werden bei Flex-Applikationen ebenfalls in MXML beschrieben. Im Fall des Metamodell-Browsers wurde eine *Transition* in Form eines Fading-Effekts für den Wechsel zwischen den beiden States sowie *Glow*-Effekte hinzugefügt (Anhang Listing 12). Die Glow-Effekte sollten ein ausgewähltes Element zusätzlich hervorheben.

### 8.2.2 Benutzerdefinierte Flex-Komponenten

Für den Metamodell-Browser wurden im Endeffekt drei verschiedene benutzerdefinierte Flex-Komponenten entwickelt. Zum einem waren dies die grafischen Container für die Elemente und Relationen, wie bereits im Feinkonzept beschrieben, zum anderen wurde eine zusätzliche graphische Komponente (*Comp_RelationLine*) für die Darstellung der Relation zwischen zwei Elementen entwickelt. Diese Komponente besteht aus einem Pfeil mit Bezeichnungstext, der die Beziehung zwischen zwei Elementen visualisiert. Der Vorteil eigenständiger Komponenten liegt darin, dass sie zur Laufzeit instanziiert werden können und die Daten der

einzelnen Objekte an die Komponenten gebunden werden. Die benutzerdefinierten Komponenten entsprechen somit graphischen Darstellungen der instanziierten Java-Klassen, die in der serverseitigen Java-Logik aus dem XML-File generiert wurden.

Exemplarisch wird in Folge eine der benutzerdefinierten Komponenten genauer betrachtet, die Komponente *Comp_RelationLine* (Abbildung 44 + Listing 6).

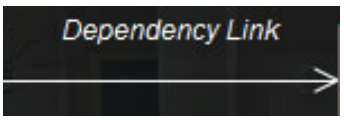


**Abbildung 44: Komponente Comp_RelationLine**

```
<s:BorderContainer xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx" width="150" height="70"
      x="{this.x}" y="{this.y}" backgroundAlpha="0.0"
      borderVisible="false" initialize="init()" doubleClickEnabled="true"
      click="showRelation(); showRelationAttributes();
      ShowRelationsElements();" mouseDownEffect="{glowElement}"
      mouseUpEffect="{unglowElement}">



      <fx:Declarations>
            <!-- Platzieren Sie nichtvisuelle Elemente (z. B. Dienste,
            Wertobjekte) hier -->
            <s:Fade id="myFadeIn" alphaFrom="0" alphaTo="1" duration="1500"
            target="{BC_Linie_Relation}"/>
            <mx:Glow id="glowElement" duration="100"
                        alphaFrom="0" alphaTo="1"
                        blurXFrom="0.0" blurXTo="30.0"
                        blurYFrom="0.0" blurYTo="30.0"  strength="2"
                        color="0xCCFFCC" />
            <mx:Glow id="unglowElement" duration="800"
                        alphaFrom="1" alphaTo="0"
                        blurXFrom="30.0" blurXTo="0.0"
                        blurYFrom="30.0" blurYTo="0.0"  strength="2"
                        color="0xCCFFCC" />
      </fx:Declarations>


      <s:BorderContainer id="BC_Linie_Relation" width="150" height="70"
      backgroundAlpha="0.0" borderVisible="false">
            <s:Label x="0" y="8" width="150" height="30"
```

```
            id="LBL_RelationLine" text="{relationLineText}"
            color="#FFFFFF" fontSize="12" fontStyle="italic"
            textAlign="center"/>
      <s:Line id="LINE_Relation" xFrom="0" xTo="150" yFrom="35"
            yTo="35">
            <s:stroke>
                  <s:SolidColorStroke color="#FFFFFF" weight="1"
                        alpha="1.0"/>
            </s:stroke>
      </s:Line>
      <s:Line id="LINE_RelationArrowLeft" xFrom="150" xTo="140"
            yFrom="35" yTo="30">
            <s:stroke>
                  <s:SolidColorStroke color="#FFFFFF" weight="1"
                        alpha="1.0"/>
            </s:stroke>
      </s:Line>
      <s:Line id="LINE_RelationArrowRight" xFrom="150" xTo="140"
            yFrom="35" yTo="40">
            <s:stroke>
                  <s:SolidColorStroke color="#FFFFFF" weight="1"
                        alpha="1.0"/>
            </s:stroke>
      </s:Line>
<fx:Script>
      <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.containers.TitleWindow;
            import mx.controls.Alert;
            import mx.core.Application;
            import mx.core.FlexGlobals;
            import mx.core.IFlexDisplayObject;
            import mx.events.CloseEvent;
            import mx.managers.PopUpManager;

            import spark.components.Application;


      [Bindable] public var relationLineId:int;
      [Bindable] public var relationLineText:String;
      [Bindable] public var relationLineFromClass:String;
      [Bindable] public var relationLineToClass:String;
```

```
            [Bindable] public var relationLineAttribute:ArrayCollection;


            private var dataRelationText:String;
            private var dataRelationAttribute:ArrayCollection;


            private function init():void {
                  myFadeIn.play([BC_Linie_Relation]);
            }


            private function showRelation():void {
            FlexGlobals.topLevelApplication.showRelation(relationLineId,
            relationLineText);
            }


            private function showRelationsElements():void {
     FlexGlobals.topLevelApplication.showRelationElements(relationLineText,
       relationLineFromClass, relationLineToClass, relationLineAttribute);
                              }


            private function showRelationAttributes():void {
       FlexGlobals.topLevelApplication.SCROLL_SelRelWithEl.viewport.vertical
       ScrollPosition = 0;
    FlexGlobals.topLevelApplication.showRelationAttributs(relationLineText,
       relationLineAttribute);
                              }
            ]]>
       </fx:Script>
       </s:BorderContainer>
</s:BorderContainer>
```

**Listing 6: Quellcode der Komponente Comp_RelationLine**

Ausgangspunkt für eine benutzerdefinierte Komponente ist eine der Standardkomponententypen der Flex-Bibliothek. Bei *Comp_RelationLine* handelt es sich dabei um einen gewöhnlichen *BorderContainer*, dessen Transparenz den Wert 0 (backgroundAlpha="0.0") hat, da bei der fertigen Komponente nur der Pfeil und die Schrift sichtbar sein sollen. Im *Declarations*-Tag werden nichtvisuelle Elemente definiert. In diesem Fall ein *Fading*-Effekt beim Einblenden der Komponente bzw. zwei aufeinanderfolgende *Glow*-Effekte, wenn ein Mausklick auf die Komponente ausgeführt wird. Der

Bezeichnungstext besteht aus einer einfachen *Label*-Komponente, deren Wert, also der angezeigte Text, über Data-Binding an die Bezeichnungsvariable (in diesem Fall *relationLineText*) aus dem ActionScript-Objekt gebunden ist. Der Pfeil wurde aus drei *Line*-Komponenten, durch die Angabe von Anfangs- und Endpunkten, zusammengestellt.

Die Funktionalität der Komponente wird innerhalb des *fx:Script*-Tag mit ActionScript definiert. Bei der Initialisierung der Komponente wird die init()-Funktion aufgerufen. Diese löst den zuvor erwähnten Fading-Effekt aus, der die Komponente am Bildschirm langsam erscheinen lässt. Per Mausklick auf die Komponente werden nacheinander die drei weiteren Funktionen ausgeführt; *showRelation(), showRelationElements()* und *showRelationAttributes*. Diese Methoden sind in weiterer Folge dafür zuständig, dass ein Seitenwechsel auf die Relationsansicht erfolgt, zu der entsprechenden Relationskomponente gescrollt wird und dass alle Attribute der Relation angezeigt werden.

### 8.2.3 ActionScript Objekt-Klassen und GUI-Logik

Jede Java-Objektklasse auf dem Server benötigt in einer Flex-Applikation eine ActionScript-Objektklasse mit genau demselben Aufbau (Anhang Listing 13 (vgl. Anhang Listing 7)). Auch das Paket, in dem die Objektklassen liegen, muss mit dem der Java-Objektklassen übereinstimmen, dies gilt auch für den Pfad zum Paket.

Besondere Bedeutung kommt dieser Zeile zu:

```
[RemoteClass(alias="metamodell.domain.Relation")]
```

Sofern eine ActionScript-Klasse einer serverseitigen Java-Klasse nachgebildet wird, so wie es hier der Fall ist, muss die Original-Klasse mitsamt dem dazugehörenden Paketnamen in der ActionScript-Klasse angegeben werden. Dadurch kann BlazeDS zwischen dem Java-Objekt und demActionScript-Objekt eine Verlinkung herstellen.

Die GUI-Logik, die in ein eigenes ActionScript-File (MMB_Modul.as) ausgelagert wurde, beinhaltet die gesamte Funktionalität der Flex-Benutzeroberfläche und wird ausschließlich client-seitig ausgeführt. In ihr sind sämtliche Funktionen enthalten, welche die Daten, die vom Server abgerufen werden, für eine grafische Darstellung aufbereiten und die eine Interaktion des Benutzers mit der Applikation ermöglichen.

Zum Beispiel führt ein Mausklick auf einen Eintrag in der Tree-Komponente zum Aufruf der Funktion *onSelectMetamodel()*. Die Funktion schreibt den gewählten Wert in eine Variable,

ruft remote die Server-Methode *getElementsFromMetamodel()* auf und übergibt den Metamodell-Namen.

Sobald ein Resultat auf diesen Remote-Call vom Server empfangen wurde, wird die Funktion *getElementsFromMetamodelCallback()* (Anhang Listing 14) ausgeführt. Diese Funktion instanziiert anhand der Menge von Objekten, die vom Server zurückkommen, für die grafische Darstellung genauso viele benutzerdefinierte Flex-Komponenten vom Typ *Comp_Element*. Über eine if-Abfrage wird entschieden, ob es sich um ein konkretes oder ein abstraktes Element handelt. Unterschieden wird aufgrund der Konvention, dass jedes abstrakte Element mit einem __ (Doppelunterstrich) beginnt. Dadurch wird den instanziierten Komponenten eine Farbe zugeteilt und sie werden in den dafür vorgesehenen *BorderContainer* für die grafische Darstellung abgelegt.

Weitere Funktionen dienen dazu, dem Benutzer möglichst viele Informationen über das gewählte Metamodell zur Verfügung zu stellen und Zusammenhänge unter den Elementen zu veranschaulichen. So wird ein Mausklick auf eine grafische Element-Komponente bzw. auf einen Eintrag in der Element-Liste immer dazu führen, dass die Accordion-Komponente in die Elemente-Ansicht wechselt, das gewählte Element angewählt wird, dessen Attribute aufgelistet  und die Beziehungen zu anderen Elementen grafisch dargestellt werden. Das alles wird noch zusätzlich durch visuelle Effekte untermalt wie zum Beispiel durch den Scroll-Effekt, der die gewünschte Komponente aus dem *BorderContainer*, in dem die Komponente hinterlegt ist, heraussucht.

Für die Arbeitsweise der weiteren Funktionen, sei an dieser Stelle auf den Source-Code der Applikation verwiesen, in denen alle Funktionen dokumentiert sind.

### 8.2.4 Preloader

Um Flex-Anwendungen mit Hilfe des FlashPlayer-PlugIns im Browser auszuführen, müssen diese zuerst einmal vom Server heruntergeladen werden. Je nach Umfang und Größe der Applikation kann es mitunter zu einer längeren Wartezeit für den Benutzer kommen, in der auf dem Bildschirm nicht viel passiert. Im schlechtesten Fall zeigt der Bildschirm eine rein weiße Oberfläche. Um dem Benutzer zu zeigen, dass gerade eine Aktion durchlaufen wird, nämlich der Download der Applikation auf den Desktop, wurde für den Metamodel-Browser ein s.g. Preloader implementiert (Abbildung 45).

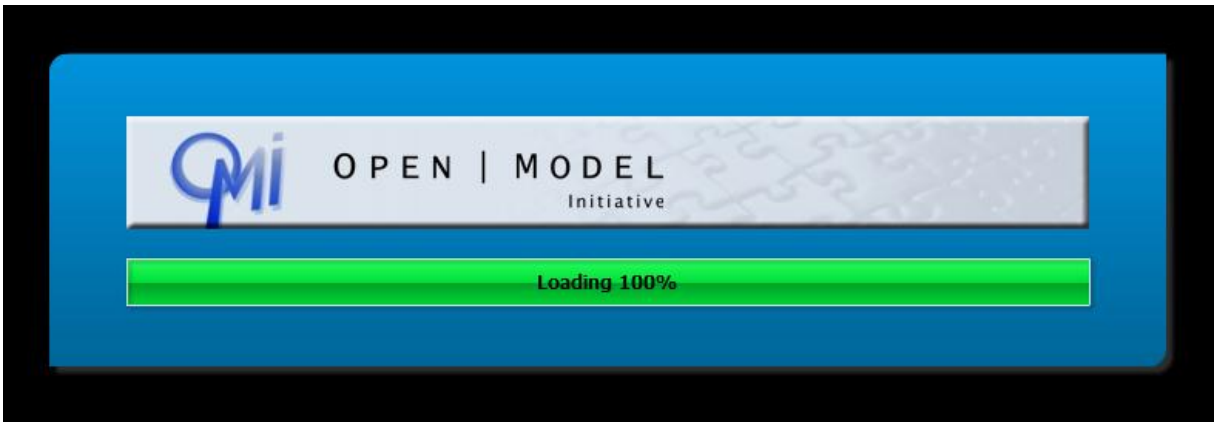


**Abbildung 45: Preloader**

Dieser zeigt den Ladevorgang der Applikation an und verkürzt so, zumindest subjektiv, die Wartezeit für den Benutzer.

Für die Einbindung eines Preloaders bietet Flex im *s:Application*-Tag ein eigenes Attribut *preloader*. In diesem Attribut wird der relative Dateipfad für die Preloader-Datei angegeben. Im Fall des Metamodel-Browsers:

preloader="preloader.CustomPreloader"

# 9 Funktion des Metamodel-Browsers (User Guide)

Dieses Kapitel beschreibt die Funktion des Metamodel-Browsers und wie ein Benutzer mit der Applikation interagieren kann, um Informationen über Metamodelle zu erlangen. Die folgenden Aufzählungen entsprechen den Markierungen in den Abbildungen.

## 9.1 Startseite

Sobald ein Benutzer die Anwendung mit einem herkömmlichen Webbrowser durch Aufruf der Internetadresse startet, wird ein Informationsbildschirm (Abbildung 46) angezeigt.

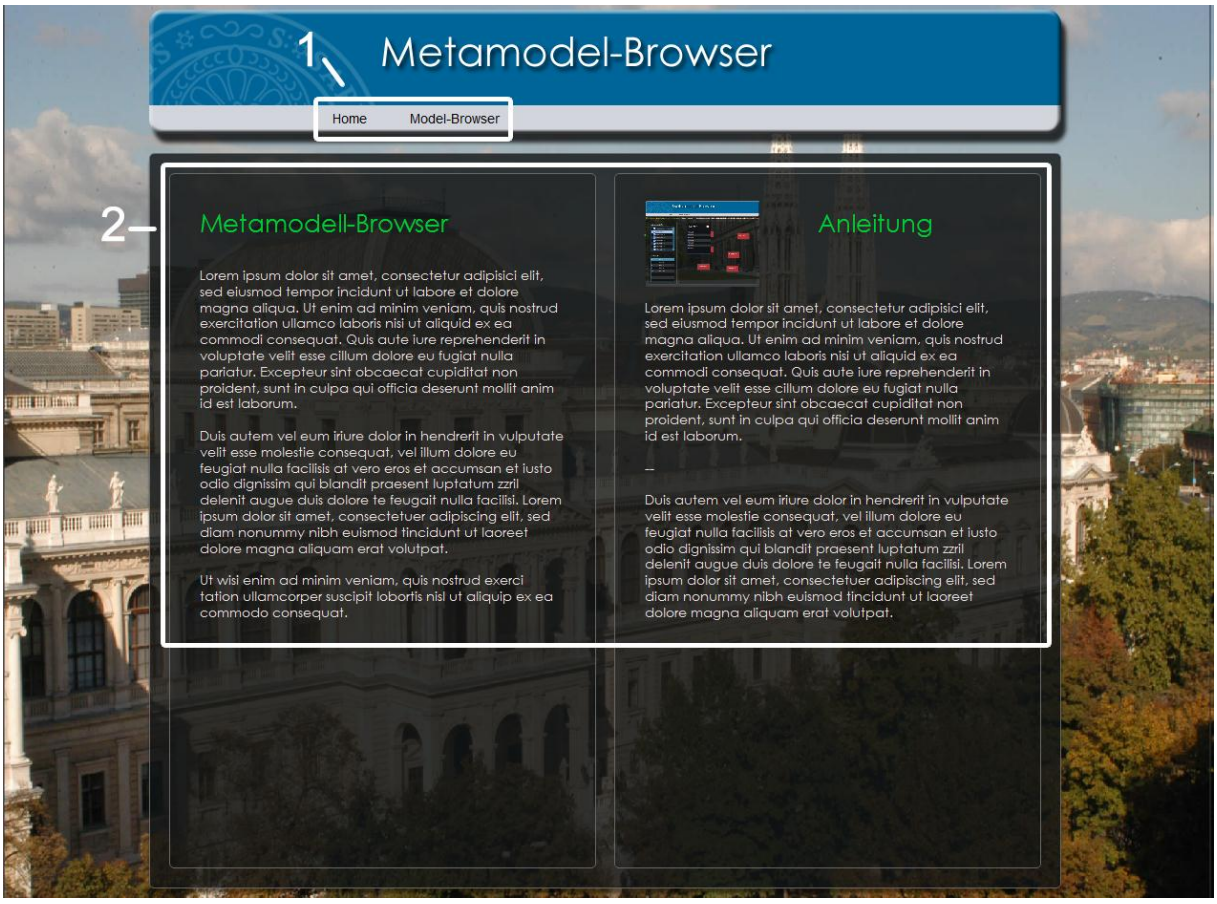


**Abbildung 46: Startseite**

1. In der Header-Leiste befindet sich das Menü. Dieses entspricht in der Funktion der Menüführung von herkömmlichen Web-Seiten. Per Mausklick kann der Benutzer zwischen der Startseite (Home) und der Metamodell-Darstellungsseite (Model-Browser) wählen.

2. In dieses Feld können Begrüßungs-, Beschreibungs- und Anleitungstexte eingefügt werden. User, die den Metamodel-Browser zum ersten Mal benützen, können sich hier einen ersten Eindruck über die Funktionsweise der Anwendung holen.

## 9.2 Metamodellseite

Sobald der Benutzer auf den Modell-Browser-Link im Menü klickt, wechselt die Applikation auf jene Seite, in der die Metamodelle angezeigt werden. Diese Seite des Browsers beinhaltet drei Ansichten, in denen die verschiedenen Komponenten des Metamodells genauer betrachtet werden können, bzw. eine Gesamtübersicht grafisch dargestellt wird. Es sind dies die Element-Ansicht, die Relationen-Ansicht und die Overview-Ansicht.

### 9.2.1 Elemente-Ansicht

In dieser Ansicht werden dem Benutzer die Details der einzelnen Elemente des Metamodells angezeigt. So erhält er Informationen über die Attribute der einzelnen Elemente oder auch darüber, in welcher Beziehung ein Element zu einem anderen Element stehen kann.

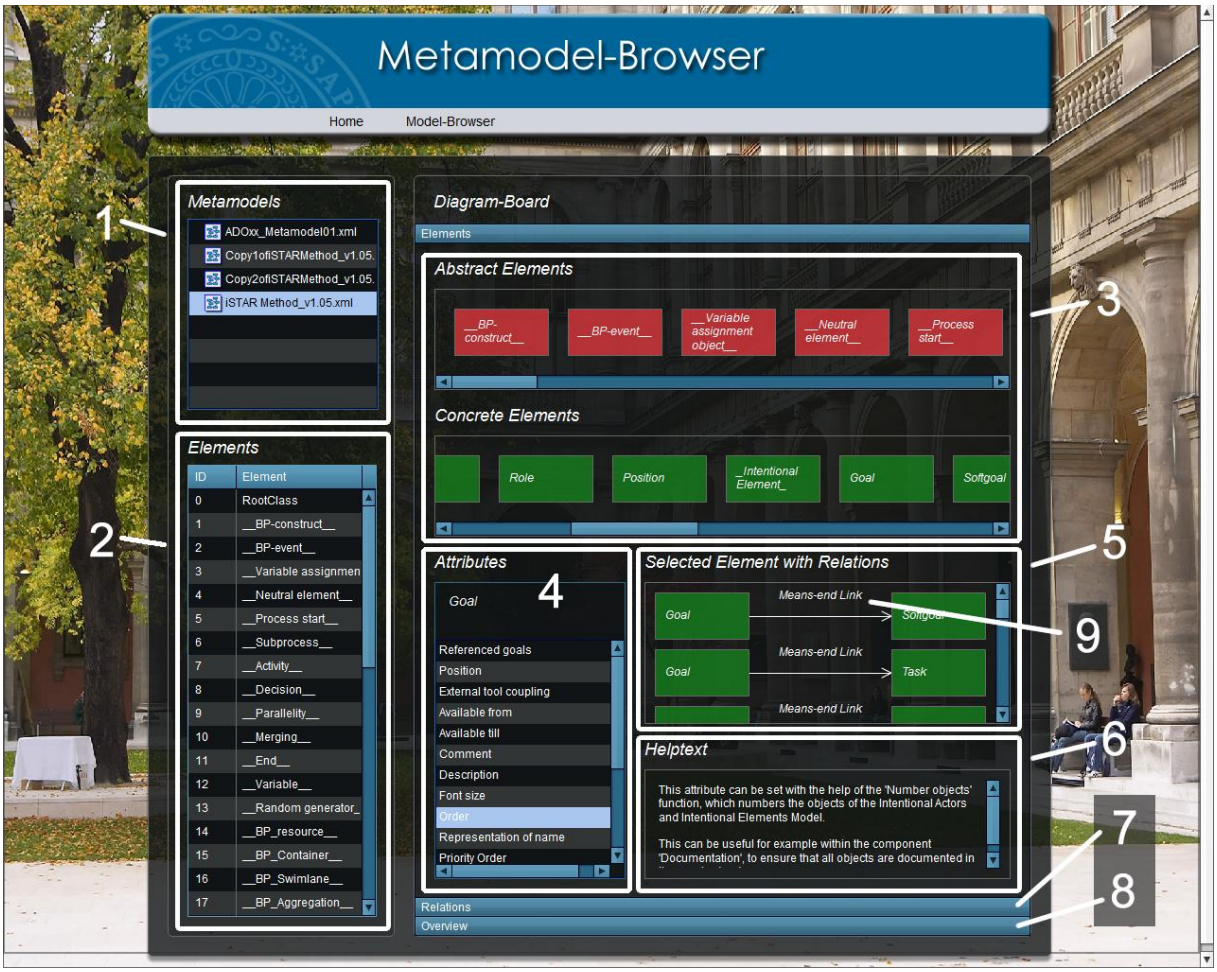


**Abbildung 47: Elemente-Ansicht**

1. Sobald die Seite erscheint, werden die zur Verfügung stehenden Metamodelle in der Listenkomponente links oben angezeigt. Per Mausklick kann der Benutzer eines der Metamodelle für eine genauere Betrachtung auswählen. Zu diesem Zeitpunkt befindet sich die *Accordion*-Komponente in der *Elements*-Ansicht.
2. Hat sich der Benutzer für ein Metamodell entschieden, werden in der Komponente links unten alle Elemente aufgelistet, d. h. sowohl abstrakte als auch konkrete Elemente. Diese Liste ermöglich jederzeit ein schnelles Navigieren zu einem Element, da sie ständig angezeigt wird.
3. Gleichzeitig mit der Liste unter Punkt 2 werden diese beiden BorderContainer mit graphischen Darstellungen der Elemente gefüllt. Im Gegensatz zur Elementenliste werden sie aber in dieser Ansicht in abstrakte und konkrete Elemente unterteilt. Würde man zu diesem Zeitpunkt in der Liste 2 auf ein Element klicken, so würde in Ansicht 3 zu dem gewünschten Element gescrollt und das Element mit einem Glow-Effekt dargestellt werden.
4. Klickt der Benutzer nun auf eines der graphischen Symbole, so werden an dieser Stelle sämtlich Attribute des Elements aufgelistet. Ein weiterer Mausklick auf eines der Attribute lässt einen Beschreibungstext des Attributs in Feld 6 (Helptext) erscheinen.
5. Gleichzeitig mit den Attributen in Feld 4 werden in Feld 5 das ausgewählte Element sowie alle Relationen zu anderen Elementen angezeigt. Klickt man auf eines der Elemente, werden umgehend dessen Attribute in Feld 4 angezeigt.
6. Dieses Feld beinhaltet den schon unter Feld 4 erwähnten Beschreibungstext für die einzelnen Attribute eines Elements.
7. Über diesen Reiter der *Accordion*-Komponente gelangt man direkt zur Relationen-Ansicht.
8. Dementsprechend gelangt über diesen Reiter zur grafischen Gesamtübersicht des Metamodells.
9. Klickt man in Feld 5 direkt auf eine der angezeigten Relationen, wechselt die Applikation automatisch in die Relationen-Ansicht und scrollt zu jenem graphischen Symbol, welches die Relation repräsentiert. Gleichzeitig werden die Attribute der Relation angezeigt sowie alle Elemente, zwischen denen diese Beziehung gelten kann.

### 9.2.2 Relationen-Ansicht

Diese Ansicht ist das Pendent zur Elemente-Ansicht, mit dem Unterschied, dass hier die **Beziehungen** des Metamodells im Mittelpunkt stehen (Abbildung 48).

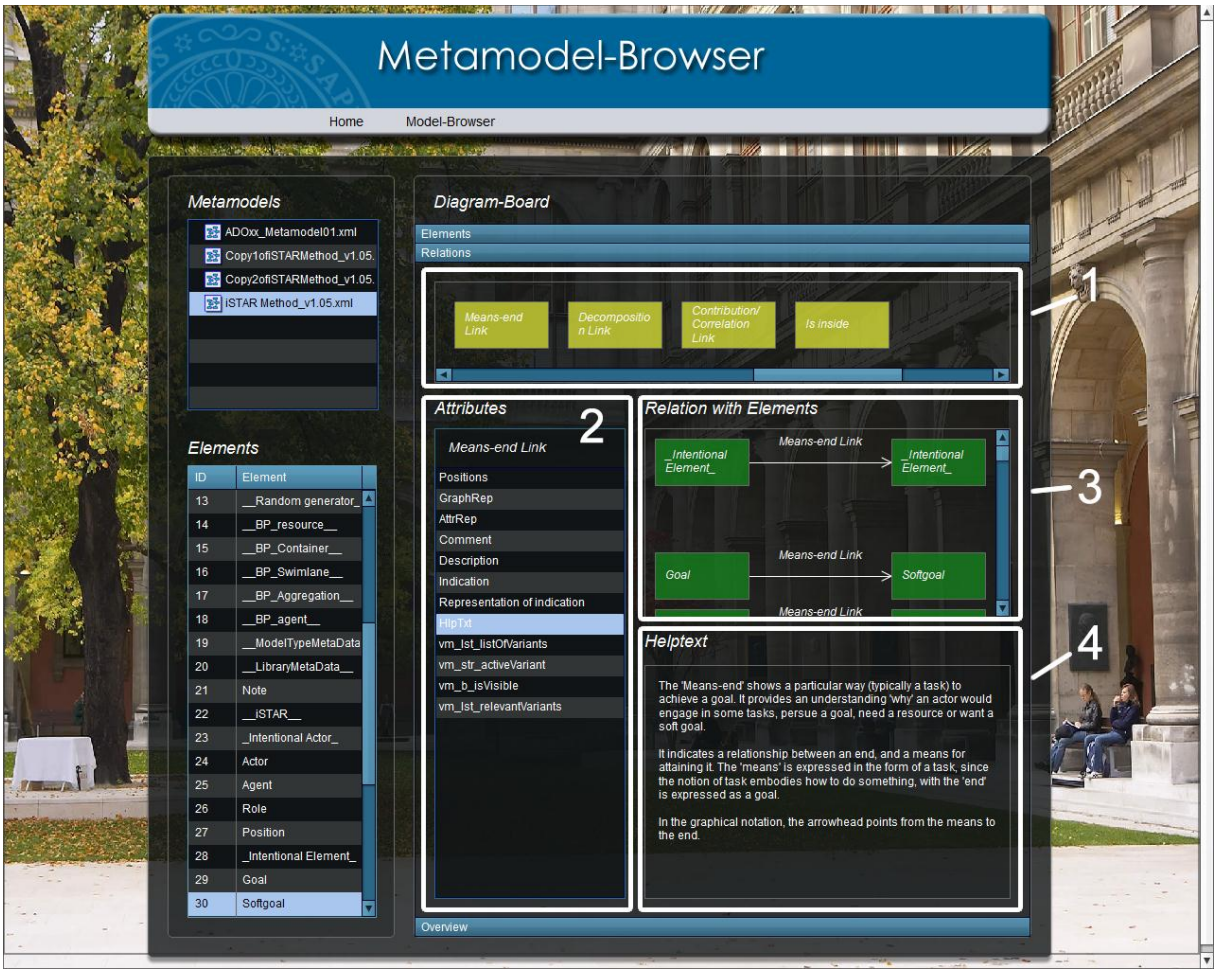


**Abbildung 48: Relationen-Ansicht**

1. Feld 1 beinhaltet alle Relationen als graphische Komponenten in einem BorderContainer. Sobald eine Relation ausgewählt wurde, werden deren Attribute in Feld 2 aufgelistet. Gleichzeitig werden alle Elemente, die durch diese Relation in Beziehung stehen können, in Feld 3 angezeigt.
2. Genauso wie bei den Elementen kann hier ein Attribut ausgewählt werden. Sofern ein Beschreibungstext existiert, erscheint er im Feld 4.
3. Hier werden alle Kombinationsmöglichkeiten jener Elemente angezeigt, die durch diese Relation in Beziehung stehen können. Benötigt der Benutzer genauere Informationen zu einem Element, kann er dieses anklicken, die Applikation wechselt in die Elemente-Ansicht und scrollt zu der gewünschten grafischen Komponente.

4. Dieses Feld zeigt, sofern vorhanden, den Beschreibungstext eines Attributs. Ist kein Hilfetext vorhanden, wird *„No Helptext available!“* in der Textbox angezeigt.

### 9.2.3 Overview-Ansicht

Als dritte Informationsquelle über die Elemente, Eigenschaften und Beziehungen eines Metamodells dient die Gesamtansicht in Form einer Bitmap-Grafik (Abbildung 49). Sie kann über den Reiter *Overview* der *Accordion*-Komponente erreicht werden (siehe Markierung 8 in Abbildung 46).

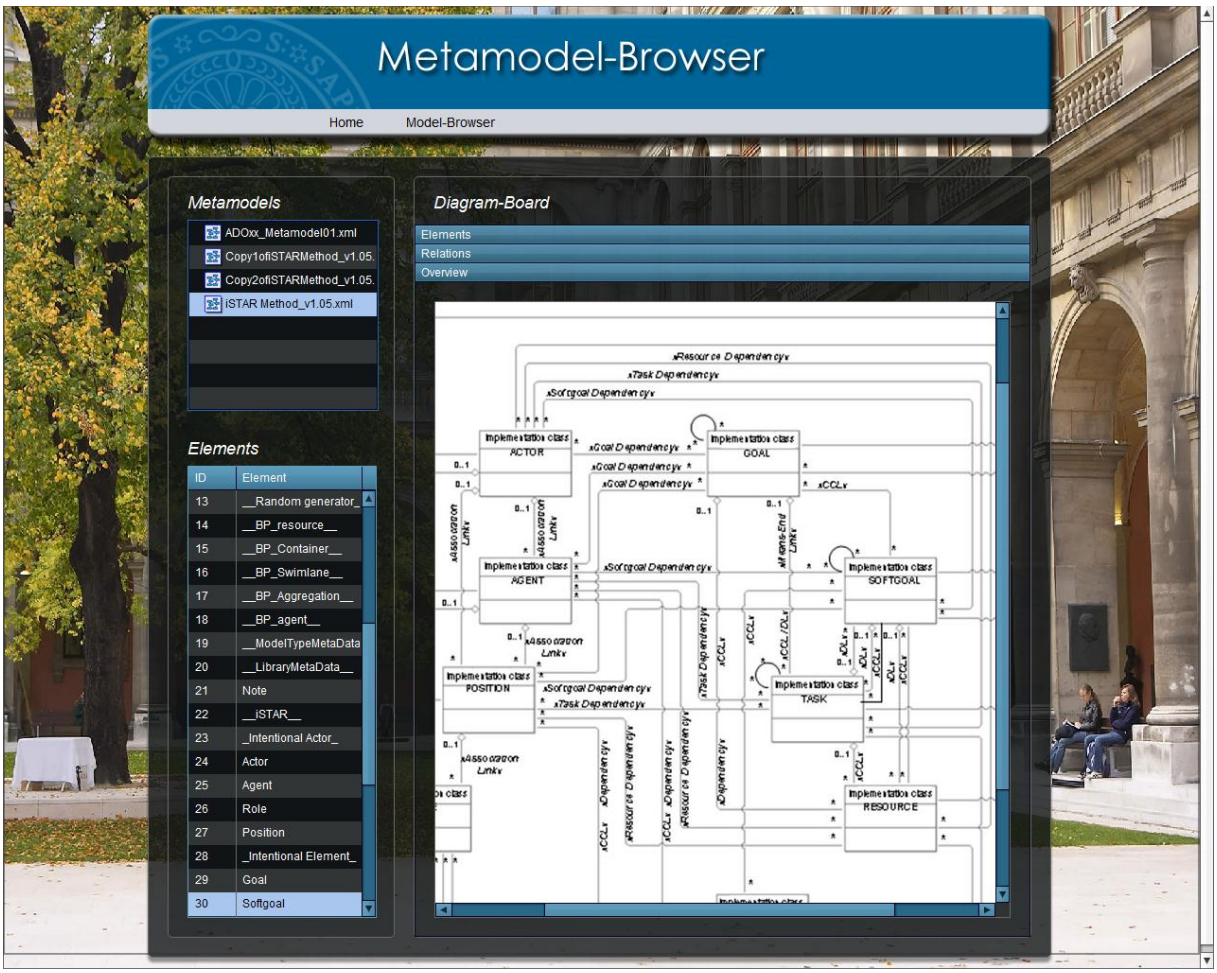


**Abbildung 49: Overview-Ansicht**

Diese Art der Gesamtdarstellung des Metamodells ist im Fall des Prototyps rein plakativ und lässt somit keine weiteren Interaktionsmöglichkeiten zu.

# 10 Zusammenfassung und Ausblick

Seit ihrer Gründung versucht die Open-Model-Initiative, dem Open-Source-Gedanken folgend, eine Plattform zu etablieren, die den freien Austausch von (Referenz-)Modellen ermöglicht. Ganz dem Motto "models for everyone" entsprechend, sollen Modelle, die bereits als Lösungen für domainspezifische Problemstellungen entwickelt wurden, allen Interessenten zur Verfügung stehen. Durch diese Wiederverwendung von Modellen könnten redundante Entwicklungen vermieden und in Folge dessen hohe Entwicklungskosten und Zeitaufwände verringert werden. Darüber hinaus soll durch die Etablierung einer Open-Model-Community Entwicklern die Möglichkeit geboten werden, diese Plattform für die Entwicklung und Diskussion von neuen Modell-Projekten in kooperativer und verteilter Arbeit zu nützen.

Stellt man den Open-Source-Gedanken dem Open-Model-Gedanken gegenüber, wird man schnell feststellen, dass es eine weitaus anspruchsvollere Herausforderung ist, eine Community für Open-Model zu begeistern, als dies bei Open-Source der Fall ist. Die Gründe dafür sind an folgenden Punkten nachvollziehbar:

- Die Nachfrage nach Open-Source ist bis dato ungleich höher als der Bedarf nach frei verfügbaren Modellen.
- Das Ausbildungsniveau von potentiellen Open-Source-Entwicklern erfordert bei weitem nicht das Wissen, welches für die Lösung von komplexen Modellierungsproblemen von Nöten ist, d. h. es ist leichter, sich als Mitglied der Open-Source-Community an einem Projekt zu beteiligen (sofern man über grundlegende Programmierkenntnisse verfügt), als sich an Modellierungsprojekten sinnvoll zu beteiligen.
- Eine vorhandene Problemstellung sollte im Normalfall bei Open-Source-Projekten um einiges leichter zu formulieren sein, als dies bei einer Modellierung der Fall ist. Das rührt daher, dass es sich bei einem Modell um ein Abbild der Realität handelt und nicht um die technische Umsetzung einer speziellen Anforderung. Diese Abstraktion der Realität beinhaltet immer eine subjektive Komponente, die zu wesentlich mehr unterschiedlicheren Interpretationen Anlass gibt, als dies bei Open-Source-Projekten wahrscheinlich ist.
- Die Sprachen, die bei Open-Source-Projekten Verwendung finden, sind vorgegeben. Es ist äußerst unwahrscheinlich, dass es für eine Problemstellung keine passende,

schon vorhandene (Programmier-)Sprache gibt, und diese somit erst formuliert bzw. entwickelt werden müsste. Bei Modellierungslösungen ist es im Gegensatz dazu sogar sehr wahrscheinlich, dass die Modellierungssprache für bestimmte Sachverhalte und Domains angepasst werden muss. Das bedeutet, dass ein Metamodell, welches die Modellierungssprache beschreibt, angepasst bzw. völlig neu entwickelt werden muss, um zu einer Lösung einer spezifischen Problemstellung zu gelangen. Letztgenannter Vorgang erfordert aber Expertenwissen und ist somit nur bedingt als Lösungsaufgabe für eine breite Community geeignet.

Der Modellbegriff an sich unterliegt schon seit Jahrtausenden unterschiedlichsten philosophischen Betrachtungen und kann daher nicht ohne weiteres auf ein rein technisches Niveau herunter gebrochen werden. Dadurch war es auch ein wichtiger Teil dieser Arbeit, einen erkenntnistheoretischen Bezugsrahmen zu erstellen. Dieser Bezugsrahmen führt dem Leser vor Augen, wie komplex der Prozess der Erkenntnis ist und wie schwierig es ist, zu einem gemeinsamen Modellierungsverständnis über einen abzubildenden Realsachverhalt zu gelangen. Die Erkenntnisse aus diesen Analysen hatten aber auch direkten Einfluss auf die Gestaltung des Metamodell-Browsers. Es wäre an der Intention der Applikation vorbeigegangen, eine völlig neue, innovative Darstellungsform zu wählen, die von den wenigsten Benutzern intuitiv zu interpretieren gewesen wäre. Darüber hinaus musste gewährleistet werden, dass die Interaktion mit der Applikation zu einem raschen Informationsgewinn führt. Der Gedanke, der dem **experimentellen Entwurf** zugrunde lag, durch möglichst viel Interaktion und kreativen Einsatz des Benutzers dessen Interesse an der Modellierungs-Thematik zu wecken, wurde deshalb zugunsten einer konservativeren bzw. intuitiveren Darstellungs- und Interaktionsform wieder verworfen.

Das Ergebnis dieser Arbeit ist somit die prototypische Entwicklung eines Metamodell-Browsers, der einerseits Metamodelle in einer visuell anspruchsvollen Form als Rich-Internet-Applikation darstellt und andererseits, aufgrund der durchgeführten Analysen, auf die Bedürfnisse der Open-Model-Community abgestimmt ist.

Obwohl der Metamodell-Browser in der derzeitigen Version bereits einsetzbar ist, bietet er noch großes Potential für Erweiterungen.

- Entwicklung eines Algorithmus zur grafischen Darstellung des gesamten Metamodells: In der jetzigen Version der Applikation wird eine Gesamtübersicht des Metamodells nur in Form einer Bitmap-Graphik angeboten. Diese lässt aber keine

weiteren Interaktionsmöglichkeiten zu. Würde die Übersicht direkt aus den graphischen Komponenten (die Elemente sowie deren Relationen untereinander) generiert, so könnte man sich auch aus der Übersichtsgraphik Detailinformationen zu den einzelnen Komponenten anzeigen lassen.

Mögliches Szenario: Sobald der Benutzer per Mausklick ein Element in der Übersichtsgraphik auswählt, wechselt die Applikation in die *Element-Ansicht*, scrollt zu dem entsprechenden Element, zeigt alle Attribute des Elements an und alle Relationen mit denen es zu anderen Elementen in Beziehung steht. Entsprechendes gilt für die Auswahl einer Beziehung und der Relationen-Ansicht.

- Ersetzen der grafischen Notation von Elementen und Relationen durch eine Metamodell-spezifische Notation: Teil einer Modellierungsmethode ist die graphische Notation der einzelnen Komponenten. Die Assoziation zu einem Metamodell könnte immens gesteigert werden, wenn die Komponenten nicht in der jetzigen Rechteckform (nur farblich unterschieden) angezeigt würden, sondern in der Notation, wie sie im Metamodell definiert ist.

  Als Beispiel könnte die Darstellung von konkreten Elementen der iStar-Modellierungsmethode statt wie bisher:

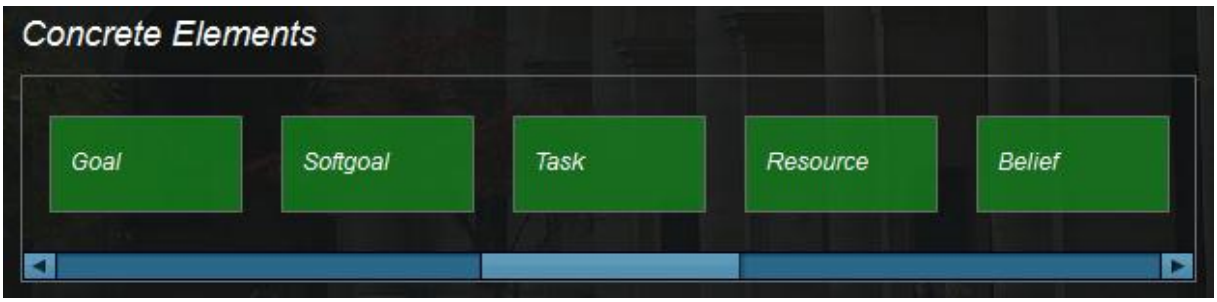


**Abbildung 50: Darstellung der konkreten Elemente (Prototyp)**

in dieser Form dargestellt werden:

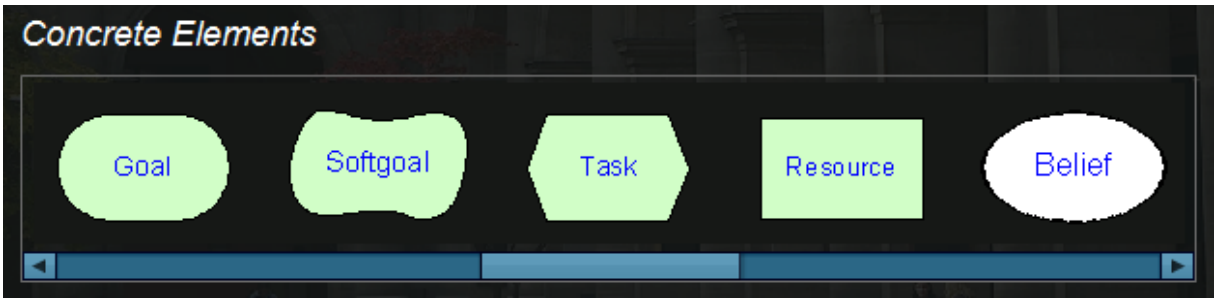


**Abbildung 51: Darstellung der konkreten Elemente (Ausblick)**

- Suchfunktion für verschiedene Notationen eines Metamodells: Indem die Notation eines Metamodells variieren kann, wäre es sinnvoll, eine Erweiterung in die Applikation zu integrieren, welche diese Alternativ-Notationen anzeigt. Dadurch könnte der Benutzer eine Notation auswählen, in der die Komponenten des Metamodells in der Applikation angezeigt werden.
- Integration eines Instant-Messangers: Ein Instant-Messanger würde den Mitgliedern der Open-Model-Community die Möglichkeit bieten, online über Metamodelle zu diskutieren. Zum Beispiel könnte nach Initiierung eines neuen Projekts durch einen Chef-Entwickler erörtert werden, inwieweit ein vorhandenes Metamodell für die Lösung einer Problemstellung geeignet wäre bzw. inwieweit ein bestehendes Metamodell angepasst oder erweitert werden müsste, um die Aufgabenstellung zu bewältigen. Insbesondere in einem kollaborativen und verteilten Entwicklungsumfeld wäre diese Erweiterung wünschenswert.
- Integration zusätzlicher Datenquellen: Im Prototyp des Metamodell-Browsers werden ausschließlich XML-Dateien im ADOxx v1.1-Format unterstützt, die direkt auf dem Server hinterlegt sind. Mit wachsender Etablierung der Open-Model-Initiative sollte auch die Möglichkeit geschaffen werden, Metamodelle, die z. B. im JSON-Format vorliegen, zu visualisieren.

  Eine weitere Möglichkeit wäre der Zugriff über Hybernate auf Metamodelle, die in Datenbanken hinterlegt sind.

  Durch die serverseitige Java-Logik und die objektorientierte Auslegung der Applikation sollte es im Bereich Datenanbindung kaum Einschränkungen geben.
- Darstellung von (Referenz)-Modellen: Der Prototyp der Applikation ist darauf ausgelegt, Metamodelle zu visualisieren. Es würde aber nichts dagegen sprechen, diese Plattform auch dahingehend zu nutzen, um bereits entwickelte (Referenz-) Modelle darzustellen bzw. anzubieten. Speziell die Möglichkeit, (Referenz-) Modelle im direkten Kontext mit den dazugehörenden Metamodellen betrachten zu können, wäre für die Lösung von neuen Modellierungsaufgaben eine große Hilfe.

Anhand dieser Auflistung wird schnell ersichtlich, dass diese Applikation ein großes Erweiterungspotential hat. Die Implementierung als Rich-Internet-Applikation gestattet es jedem Interessenten, sich Informationen über bestehende Metamodelle anzueignen und das daraus generierte Wissen im Sinne der Open-Model-Initiative weiterzuverwenden.

# Anhang

```
package metamodell.domain;

import java.util.ArrayList;

public class Relation {
	private int relationId;
	private String relationName;
	private String fromClass;
	private String toClass;
	private ArrayList<Attribut> attribute;

	public int getRelationId() {
		return relationId;
	}
	public void setRelationId(int relationId) {
		this.relationId = relationId;
	}
	public String getRelationName() {
		return relationName;
	}
	public void setRelationName(String relationName) {
		this.relationName = relationName;
	}
	public String getFromClass() {
		return fromClass;
	}
	public void setFromClass(String fromClass) {
		this.fromClass = fromClass;
	}
	public String getToClass() {
		return toClass;
	}
	public void setToClass(String toClass) {
		this.toClass = toClass;
	}
	public ArrayList<Attribut> getAttribute() {
		return attribute;
```

```
	}
	public void setAttribute(ArrayList<Attribut> attribute) {
		this.attribute = attribute;
	}
}
```

**Listing 7: Quellcode der Klasse Relation**

```
// Methode liest alle Metamodell-Namen aus dem Metamodell-Verzeichnis
public ArrayList<String> getAllMetamodelNames() {
	File directory = new File("..\\webapps\\XML_Metamodelle");
	ArrayList<String> metamodelNames = new ArrayList<String>();
	// Ueberpruefen, ob das angegebene Verzeichnis existiert
	if (!directory.exists()) {System.out.println("Das Verzeichnis " +
"existiert nicht!");
		} else {
			metamodelNames = dirListing(directory);
		}
		return metamodelNames;
}

// Liste die Metamodelle auf
public ArrayList<String> dirListing(File directory) {

	ArrayList<String> metamodellName = new ArrayList<String>();

		File[] files = directory.listFiles();

		for (int i = 0; i < files.length; i++) {
			File file = files[i];
			metamodellName.add(file.getName());
		}
	return metamodellName;
}
```

**Listing 8: Methoden getAllMetamodelNames() und dirListing()**

```
// Methode instanziiert die Relations- und Attribut-Objekte und befüllt
// sie mit den Daten aus dem XML-File
public List<Relation> getRelationsFromMetamodel(String metamodelName)
                throws ParserConfigurationException, SAXException,
IOException, XPathExpressionException {
//Instanziirung der ArrayList relation. In dieser werden die Relations-
// Objekte gespeichert und an das Flex-Frontend zurückgegeben
relation = new ArrayList<Relation>();


try {
// Erstellen einer NodeList in der alle Elemente mit dem Namen
// "relationclass" gespeichert werden
   NodeList relationLst = doc.getElementsByTagName("relationclass");


   for (int i = 0; i < relationLst.getLength(); i++) {
         Node relationNode = relationLst.item(i);
         Element relationElement = (Element) relationNode;
         String relationName = relationElement.getAttribute("name");
         String relationFromClass =
relationElement.getAttribute("fromclass");
         String relationToClass =
relationElement.getAttribute("toclass");


// Instanziieren eines neuen Relations-Objekts und befüllen mit Daten
         relat = new Relation();
         relat.setRelationId(i);
         relat.setRelationName(relationName);
         relat.setFromClass(relationFromClass);
         relat.setToClass(relationToClass);


// Instanziirung der ArrayList relationAttribut. In dieser werden die
// relationsAttribut-Objekte gespeichert und in das Objekt der Relation
// geschrieben
         relationAttribut = new ArrayList<Attribut>();


// Erstellen einer NodeList in der alle Elemente mit dem Namen
// "attributes" gespeichert werden
         NodeList relationAttributesLst = relationElement
                     .getElementsByTagName("attributes");
         for (int j = 0; j < relationAttributesLst.getLength(); j++) {
               Element relationAttributesElement = (Element)
relationAttributesLst.item(j);
```

```
                NodeList relationAttributLst = relationAttributesElement
                            .getElementsByTagName("attribute");

                for (int k = 0; k < relationAttributLst.getLength(); k++)
{
                      Node attributNode = relationAttributLst.item(k);
                      Element attributElement = (Element) attributNode;
.
.
.
.
// for-Schleifen, welche die Attribut-Inhalte auslesen
.
.
.

   String attributName = attributElement.getAttribute("name");
   String attributType = attributElement.getAttribute("type");

// Instanziieren eines neuen Attribut-Objekts und befüllen mit den
// Daten, welche aus dem DOM-Objekt gelesen wurden
          Attribut attr = new Attribut();
          attr.setAttributId(k);
          attr.setAttributName(attributName);
          attr.setAttributType(attributType);
          attr.setAttributHilfeText(attributHilfeText);

// Hinzufügen des Attribut-Objekts in die ArrayList "relationAttribut"
          relationAttribut.add(attr);
                }
          }
// Die ArrayList der Attribut-Objekte in das Relations-Objekt schreiben
          relat.setAttribute(relationAttribut);
// Relations-Objekt in ArrayList "relation" schreiben
          relation.add(relat);
   }
          } catch (Exception e) {
                e.printStackTrace();
          }
```

```
   // Rückgabe der ArrayList "relation" an das Flex-Frontend
            return relation;
      }
   }
```

**Listing 9: getRelationsFromMetamodel – Methode**

```
// Methode speichert Übersichtsgrafik in ein ByteArray und übergibt es an
das Flex-Frontend
public byte[] getImageFromMetamodel(String metamodelName) {
      metamodelName = metamodelName.substring(0, metamodelName.length()-4);


      File metamodelImage = new File("..\\webapps\\Metamodell_Overview\\" +
            metamodelName + "_overview.jpg");
       FileInputStream fis;
        byte[] imageData =null;
        FileChannel fc;


        try {
            fis = new FileInputStream(metamodelImage);
            fc = fis.getChannel();
            imageData = new byte[(int)(fc.size())];
            ByteBuffer bb = ByteBuffer.wrap(imageData);
            fc.read(bb);
        } catch (FileNotFoundException e) {
            System.out.println("Es ist folgender Fehler aufgetreten: " +
                  e.getMessage());
        } catch (IOException e) {
            System.out.println("Es ist folgender Fehler aufgetreten: " +
                  e.getMessage());
        }
        return imageData;
      }
```

**Listing 10: Methode getImageFromMetamodel**

```
package metamodell.services;

import java.util.ArrayList;
import java.util.List;

import metamodell.domain.Klasse;
import metamodell.domain.Relation;

public interface IMetaService {

	public ArrayList<String> getAllMetamodelNames();
	public List<Klasse> getKlassenFromMetamodell(String metamodellName);
	public List<Relation> getRelationsFromMetamodell(String
metamodellName);
	public byte[] getImageFromMetamodel(String metamodellName);
}
```

**Listing 11: Java - Interface**

```
<!-- Definition der Transition zwischen zwei States -->
<s:transitions>
	<s:Transition fromState="*" toState="*" autoReverse="true">
		<s:Parallel>
			<s:Parallel targets="{[Home, inhalt_home, Applikation,
inhalt_applikation]}">
				<s:Fade duration="2000"/>
			</s:Parallel>
		</s:Parallel>
	</s:Transition>
</s:transitions>

<!-- Definition eines Golw-Effekts -->
<mx:Glow id="glowBC_Glow" duration="100"
		alphaFrom="0" alphaTo="1"
		blurXFrom="0.0" blurXTo="30.0"
		blurYFrom="0.0" blurYTo="30.0"  strength="2"
		color="0xCCFFCC" target="{BC_Glow}"
startDelay="{BC_GlowDelayTime + 100}"/>
```

**Listing 12: Definition von Transition und Glow-Effekt**

```
package metamodell.domain
{
	import mx.collections.ArrayCollection;

	[RemoteClass(alias="metamodell.domain.Relation")]
	public class Relation
	{
		[Bindable]
		public var relationId:int;
		public function getRelationId():int {
			return relationId;
		}
		public function setRelationId(value:int):void {
			relationId = value;
		}

		[Bindable]
		public var relationName:String;
		public function getRelationName():String {
			return relationName;
		}
		public function setRelationName(value:String):void {
			relationName = value;
		}

		[Bindable]
		public var fromClass:String;
		public function getFromClass():String {
			return fromClass;
		}
		public function setFromClass(value:String):void {
			fromClass = value;
		}

		[Bindable]
		public var toClass:String;
		public function getToClass():String {
			return toClass;
		}
```

```
            public function setToClass(value:String):void {
                  toClass = value;
            }

            [Bindable]
            public var attribute:ArrayCollection = new ArrayCollection();
            public function getAttribute():ArrayCollection {
                  return attribute;
            }
            public function setAttribute(value:ArrayCollection):void {
                  attribute = value;
            }
      }
      }
}
```

**Listing 13: ActionScript-Objektklasse Relation**

```
//Callback-Handler: nimmt die Elemente als Ergebnis entgegen und erstellt
die Element-Komponenten.
private function getElementsFromMetamodelCallback(event:ResultEvent):void
{
      elements =
(ArrayCollection)(MetamodelService.getElementsFromMetamodel.lastResult);
      var abstract_xValue:int = 20;
      var concrete_xValue:int = 20;
      var abstractIndex:int = 0;
      var concreteIndex:int = 0;

      BC_Abstract_Elements_width = 0;
      BC_Concrete_Elements_width = 0;
      for(var i:int = 0; i < elements.length; i++) {
            componentElement = new Comp_Element ();
            componentElement.elementId = elements[i].elementId;
            componentElement.elementText = elements[i].elementName;
            componentElement.elementSuperklasse =
            elements[i].elementSuperklasse;
            componentElement.elementsAttribute = elements[i].attribute;

            if(componentElement.elementText.substr(0, 2) == "__") {
```

```
                componentElement.CANVAS_elementBgColor = "0xDC383C";
                elements[i].elementType = "abstract";
                elements[i].elementIndex = abstractIndex;
                componentElement.elementType = "abstract";
                componentElement.elementIndex = abstractIndex;
                componentElement.x = abstract_xValue;
                componentElement.y = 20;
                BC_Elemente.addElement(componentElement);
                abstract_xValue += 120;
                abstractIndex ++;
                BC_Abstract_Elemente_width = abstract_xValue + 470;
            } else {
                componentElement.CANVAS_elementBgColor = "0x17811F";
                elements[i].elementType = "concrete";
                elements[i].ellementIndex = concreteIndex;
                componentElement.elementType = "concrete";
                componentElement.elementIndex = concreteIndex;
                componentElement.x = concrete_xValue;
                componentElement.y = 20;
                BC_Concrete_Elemente.addElement(componentElement);
                concrete_xValue += 120;
                concreteIndex ++;
                BC_Concrete_Elemente_width = concrete_xValue + 470;
            }
        }
        TABLE_Elements.dataProvider = elements;
}
```

**Listing 14: ActionScript-Funktion getElementsFromMetamodelCallback()**

# Literaturverzeichnis


[ALLAIRE2002] J. Allaire, *Macromedia Flash MX-A next-generation rich client,* White Paper der Fa. Macromedia, 2002.

[BECKNIEHKNACK04] J. Becker, B.Niehaves, R. Knackstedt. *Bezugsrahmen zur epistomologischen Positionierung der Referenzmodellierung*, in: Referenzmodellierung, S 1-17, Heidelberg 2004.

[BECKSCH2004] J. Becker, R. Schütte. *Handelsinformationssysteme*. Frankfurt/Main 2004.

[CAMPSTOCK2009] C. A. Campbell, J. Stockton. *Microsoft Silverlight 2 im Einsatz.* München 2009.

[DELFMANN2006] P. Delfmann. *Adaptive Referenzmodellierung*. Berlin 2006.

[DEWSBURY2008] R. Dewsbury. *Google Web Toolkit Applications.* Boston 2008.

[ENGELS2008] G. Engels. *Modellierungssprache,* in: Enzyklopädie der Wirtschaftsinformatik, URL: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/technologien-methoden/Sprache/Modellierungssprache. Stand: 22.09.2008.

[EXTJS2011] Sencha. *Ext JS – Cross-Browser Rich Internet Application Framework* http://www.sencha.com/products/extjs/. Letzter Zugriff: 12.02.2011.

[FLEX2011] Adobe. *Flex and Java* http://www.adobe.com/devnet/flex/flex_java.html. Letzter Zugriff: 05.01.2011.

[FRIEDMANN2007] V. Friedman. *Praxisbuch Web 2.0*. Bonn 2007.

[GWT2011] Google code. Google Web Toolkit http://code.google.com/intl/de-DE/webtoolkit/. Letzter Zugriff: 05.01.2011.

[HALLFAGEN1956] A. D. Hall, E. Fagen. *Definition of System - General Systems 1*. 1956.

[HANSONTACY2007] R. Hanson, Adam Tacy. *GWT im Einsatz - AJAX-Anwendungen entwickeln mit dem Google Web Toolkit.* München-Wien 2007.

[HILLFEHLUL1989] W. Hill, R. Fehlbaum, P. Ulrich. *Organisationslehre*. Bern-Stuttgart 1989.

[ITWISSENGWT2011] ITWissen. *Google Web Toolkit* http://www.itwissen.info/definition/lexikon/GWT-Google-web-toolkit-Google-Web-Toolkit.html/. Letzter Zugriff:14.02.2011.

[JAVAFX2011] Oracle. *Develop Expressive Content with the JavaFX Platform* http://www.javafx.com/about/overview/. Letzter Zugriff: 14. 02. 2011.

[JQUERY2011] jQuery. *jQuery is a new kind of JavaScript Library* http://jquery.com/. Letzter Zugriff 14.02.2011.

[KARKÜHN2002] D. Karagiannis, H. Kühn. *Metamodelling Platforms*. In: EC-WEB 02: Proceeding of the Third International Conference on E-Commerce and Web-Technologics. London, UK: Springer-Verlag 2002.

[LAHRAY2006] B. Lahres, G Rayman. *Praxisbuch Objektorientierung*. Bonn 2006.

[LASSMANN2006] W. Lassmann. *Wirtschaftsinformatik: Nachschlagewerk für Studium und Praxis.* Heusenstamm 2006.

[LORENZ1995] K. Lorenz. *Enzyklopädie Philosophie und Wissenschaftstheorie 3*. Stuttgart-Weimar 1995.

[MARENT 1995] C. Marent. *Branchenspezifische Referenzmodelle für betriebswirtschaftliche IV-Anwendungsbereiche*. Quelle: Wirtschaftsinformatik Ausgabe Nr.: 03-1995.

[MÜLLER2009] F. Müller. *Professionelle Rich-Client-Lösungen mit Flex und Java.* München 2009.

[MÜLLER2010] F. Müller. *Flash Builder 4 und Java*. Frankfurt am Main 2010.

[RAYMOND2001] S. Raymond. *The Cathedral and the Bazaar*. Sebastopol 2001.

[SCHEER1992] A. W. Scheer. *Architektur Integrierter Informationssysteme. Grundlagen der Unternehmensmodellierung*. Berlin 1992.

[SCHÜTTE1998] R. Schütte. *Grundsätze ordnungsmäßiger Referenzmodellierung. Konstruktionstruktion konfigurations- und anpassungsorientierter Modelle*. Wiesbaden 1998.

[SCHULZE2009] A. Schulze. *Rich Internet Applikationen – Best Practices vom Core bis zu Desktop*. München 2009.

[SILARCH2011] msdn.microsoft. *Silverlight-Architektur* http://msdn.microsoft.com/de/library/bb404713(VS.95).aspx/. Letzter Zugriff: 15. 10.2010.

[STAC73] H. Stachowiak. *Allgemeine Modelltheorie*. Wien 1973.

[STEYER2009] R. Steyer. *Einstieg in JavaFX – Dynamische und interaktive Java-Applikationen mit JavaFX*. München 2009.

[TEUBNER1999] R. A. Teubner. *Organisations- und Informationssystemgestaltung – Theoretische Grundlagen und integrierte Methoden*. Wiesbaden 1999.

[WENZ2007] C. Wenz, *JavaScript und Ajax*, Bonn 2007.

[WIDJAJA2009] S. Widjaja. *Rich Internet Applications mit Flex 3*. München 2008.

[WIDJAJA2010] S. Widjaja. *Adobe Flex 4*. München 2010.

# Lebenslauf

**Persönliche Daten:**

| | |
|---|---|
| Name: | Peter Hirsch |
| Geburtsdatum, - ort: | 27.04.1971 in Wien |
| Adresse: | 2242 Prottes, Josef Seitz-Straße 35 |
| Telefon: | 0650/850 45 49 |
| Email: | peter.hirsch@airwave.at |
| Familienstand: | Ledig |
| Staatsangehörigkeit: | Österreich |
| Grundwehrdienst: | absolviert 1991 |

**Schulausbildung:**

| | |
|---|---|
| 1977 – 1981 | Volksschule Strasshof |
| 1981 – 1987 | AHS Wien und Gänserndorf |
| 1999 – 2004 | Abendschule im Fach Elektronik am TGM Wien 20. mit Matura |
| 2005 - laufend | Studium: Wirtschaftsinformatik und Kunstgeschichte an der Universität Wien |
| 2008 | Abschluss Bakkalaureat in Wirtschaftsinformatik |

**Berufsausbildung:**

| | |
|---|---|
| 1995 – 1996 | Ausbildung zum Anlagenmonteur am Berufsförderungsinstitut Siegmundsherberg mit Abschluss |

**Berufliche Weiterbildung:**

Ausbildung in Vakuumtechnik und
KE-Schweißverfahren
Staplerführerschein
Qualitätsmanagement

**Berufstätigkeit:**

| | |
|---|---|
| 1987 – 1994 | Billeteur und Filmvorführer im KIBA Elite-Center, 1010 Wien |
| 1996 – 2005 | Fa. iSi-Airbag GmbH., Anlagenbau, 1210 Wien, Scheydgasse:<br>– 3 Jahre Schichtleiter in der Produktion<br>– 3 Jahre im internen Anlagenbau<br>– 1,5 Jahre Produktions-Abteilungsleiter<br>– Poduktionstechnik |
| 2007 - 2011 | geringfügige Beschäftigung als Programmierer, Fa. EMIG-Datenverarbeitung, 1230 Wien |
| Seit März 2011 | Fa. Domoferm, 2232 Gänserndorf |

**Fremdsprachen:** Englisch in Wort und Schrift

**Führerschein:** Fahrzeugklassen A,B