FernUniversität in Hagen
Seminar in Informatik (01917, WS2009/10)
Eclipse – Projekte und Entwicklung

Model Development Tools (MDT)

Jochen Bauer
info@jochen-bauer.net

Das Projekt „Model Development Tools“ (MDT) ist Teil des „Eclipse Modeling“-Projekts (EMP) und stellt mit seinen Schwesterprojekten Werkzeuge zur Verfügung, damit modellgetrieben Software entwickelt werden kann. MDT baut hier auf dem „Eclipse Modeling Framework“ (EMF) und dem „Graphical Editing Framework“ (GEF) auf, nutzt also bereits bestehende Projekte - grafische Editoren können auf diese Weise erstellt werden. Mehr noch, aus diesen Modellen kann Programmcode in der Zielsprache gegossen werden.

MDT samt seiner Unterprojekte fokussiert sich dabei auf die Umsetzung existierender Spezifikationen der „Object Management Group“ (OMG), genauer „Business Process Modeling Notation“ (BPMN), „Object Constraint Language“ (OCL), „Unified Modeling Language“ (UML) und „Extensible Markup Language Schema Definition“ (XSD). Neben der Implementierung dieser Spezifikationen gibt es auch weitere Unterprojekte, die die nötigen Anwendungstools stellen, damit mit den Spezifikationen gearbeitet werden kann.

1 Einleitung

Die Softwareentwicklung strebt danach, die Komplexität der Softwaresysteme besser beherrschbar zu halten und so die relative Anzahl erfolgreicher Projektabschlüsse zu steigern [SW02, S. 2 ff.]. Die modellgetriebene Entwicklung derartiger Systeme ist hier einer der Kandidaten für einen Lösungsansatz. Bei dieser modellgetriebenen Strategie soll dem Eclipse-Projekt eine Schlüsselrolle zukommen, genauer dem EMP. Der Untertitel des EMPs „A Domain-Specific Language (DSL) Toolkit“ lässt schon erahnen, dass dort vor allem Hilfen zur Entwicklung von domänenspezifischen Sprachen angeboten werden.

Das EMP fasst als Container-Projekt einige Unterprojekte, die aufeinander aufbauen oder in Wechselwirkung stehen. EMF, GEF und das „Graphic Modeling Framework“ (GMF)[1] leisten beispielsweise alles Notwendige, dass grafische Editoren erstell- und nutzbar sind. Mehr noch, sie ermöglichen, dass aus den fertigen Modellen dann Programmcode in der Zielsprache wird.

MDT baut auf diese Projekte auf und regelt, dass die bestehenden Spezifikationen umsetzbar sind. Darüber hinaus werden notwendige Tools geliefert, die einen Gebrauch der

[1] Das GMF soll die beiden Projekte EMF und GEF geschickt verzahnen.

Spezifikationen erst ermöglichen. Beispielsweise gibt es die beiden MDT-Unterprojekte „UML2“ und „UML2 Tools“ , die so das Zusammenspiel aus Spezifikations- und Werkzeugprojekt widerspiegeln. MDT spezialisiert sich dabei auf das Umsetzen existierender Standards der OMG, wie UML, OCL, XSD, BPMN.

Ziel der Arbeit ist es einen Überblick zu geben, inwieweit MDT samt seiner Unterprojekte in der Lage ist, die Lösungen für die anfallenden Probleme der modellgetriebenen Entwicklung zu liefern.

2 Modellgetriebene Entwicklung

2.1 Definition und Weg

Modellgetriebene Softwareentwicklung (MDSD) will automatisch Quellcode aus domänenspezifischen Modellen generieren. Model Driven Architecture (MDA) ist dabei der Ansatz der OMG für die modellgetriebene Softwareentwicklung. MDA ist von der OMG patentiert und folglich wird als Alternativbegriff zu MDA auch Model Driven Development (MDD) für modellgetriebene Entwicklung genutzt. Bezogen auf MDT ist die genaue Abgrenzung zwischen MDSD und MDA unerheblich, beide haben das Ziel aus Modellen Programmcode automatisch zu erstellen[2].

MDD ist möglich, wenn es eine passende Modellierungssprache und passende Werkzeuge für die betreffende Domäne gibt. Dazu müssen die Modelle in Quellcode oder in andere Modelle transformierbar sein und der Quellcode muss für eine ausgereifte Plattform erzeugt werden können. Als Austauschformat für Modelle wird häufig XML Metadata Interchange (XMI) eingesetzt. Die OMG strebt dabei noch eine Standardisierung der Transformations- und Modellierungssprachen an, die Interoperabilität der Werkzeuge soll so erst einmal möglich bzw. verbessert werden [RHQ+05, S. 59].

Eine Sprache im Sinne der MDA muss zum einen die Domäne erfassen und zum anderen rigide definiert sein, damit die Modelle validier- und zu Quellcode transformierbar sind. Die UML als Universalsprache kann und will dies so natürlich nicht leisten. Allerdings kann mit Hilfe des in der UML-Version 2.0 eingeführten Elements „Profil“ der UML-Sprachschatz auf eine bestimmte Domäne zugeschnitten werden. Um am Ende bei einer domänenspezifischen Sprache zu landen, gibt es also die Option eine Universalsprache einzuschränken oder eine eigene domänenspezifische Sprache formal von Null an zu beschreiben.

[2] Vergleich von MDA und MDSD, siehe [Etz06, S. 13]

2.2 Metamodelle und deren Bedeutung für die MDD

Ein Metamodell ist ein Modell, das die darunter liegende Schicht beschreibt und die untere Schicht ist eine Instanz der darüber liegenden Schicht [RHQ+05, S. 54]. Meta kommt aus dem Griechischen und bedeutet „Über“ , Metamodell steht also für ein Übermodell, Metaschicht für Überschicht. Über bezieht sich dabei auf Schichten, also eine Metaschicht liegt eine Schicht über der aktuellen Schicht, eine Metametaschicht eine weitere Schicht über der Metaschicht. Eine einzelne Metaschicht kann man als Metamodell bezeichnen, allerdings werden alle Metaschichten gemeinsam auch als Metamodell bezeichnet. Man muss also aufpassen, von welchen Schichten nun genau gesprochen wird, wenn von einem Metamodell die Rede ist.

Am Beispiel der UML soll dieser Schichtaufbau klar gemacht werden, siehe Abbildung S. 4. Die Schicht M0 fasst die Instanzen, also das Objektgeflecht, dass der Anwender beim Benutzen der Software erzeugt und dessen Instanzen miteinander interagieren. Auf der Schicht M1 steht das vom Programmierer modellierte Anwendungssystem. Die Schicht M0 ist eine Instanz der Schicht M1. Hier wird also geklärt, welche Attribute eine Buch-Instanz der Schicht M0 hat. Für jede Schicht muss geklärt werden, wie das Modell notiert werden soll und meist wird im objektorientierten Umfeld die UML zur Beschreibung einer jeden Schicht genutzt. Auf der Schicht M2 steht die UML-Sprachbeschreibung selbst, also die Regeln, die der Systemmodellierer einzuhalten hat. Wie schon bei den darunter liegenden Schichten wird zur Beschreibung die UML genutzt. Folglich wird M2, also die UML mit der UML-Syntax beschrieben. M2 beschreibt hier etwa, dass eine 'Class' wie 'Buch' beliebig viele 'Property'-Instanzen haben darf. Auf der Schicht M3 wird die Metaschicht zur UML beschrieben, also welcher Satz an Elementen in der UML vorkommen darf, hier etwa 'Class', 'Association' und 'Generalization'. Auf der Schicht M3 ist beim OMG-Standard die Meta-Object-Facility (MOF), was mit Metaobjektmöglichkeiten eingedeutscht werden kann.

MOF und UML sind eine mögliche Basis, wie man modellgetrieben entwickeln kann [SVEH07, S. 64]. Dabei ist MOF das Metametamodell und UML das Metamodell. Durch die bereits genannten Profile kann man sich die Universalsprache UML auf die Problemdomäne zurechtschustern. Diese Herangehensweise funktioniert, hat aber den Nachteil, dass man die UML nur schlecht um Kernelemente reduzieren kann - will man dies dennoch, muss man einen anderen Weg suchen, etwa eine eigene domänenspezifische Sprache entwerfen.

Eclipse arbeitet mit einem anderen Metametamodell als MOF, nämlich Ecore. Ecore ist dem Metametamodell „Essential-MOF“ (EMOF), eine auf das Wesentliche zusammengestauchte Version von MOF, sehr ähnlich. Ecore ist Bestandteil des Eclipse-Projekts EMF. EMF lässt dann aus solchen Ecore-Modellen die Java-Implementierung oder andere Modelle generieren. Die UML konnte mit Ecore anstatt mit MOF als Metamodell beschrieben werden und so sind UML-Modelle nun auch für EMF nutzbar.

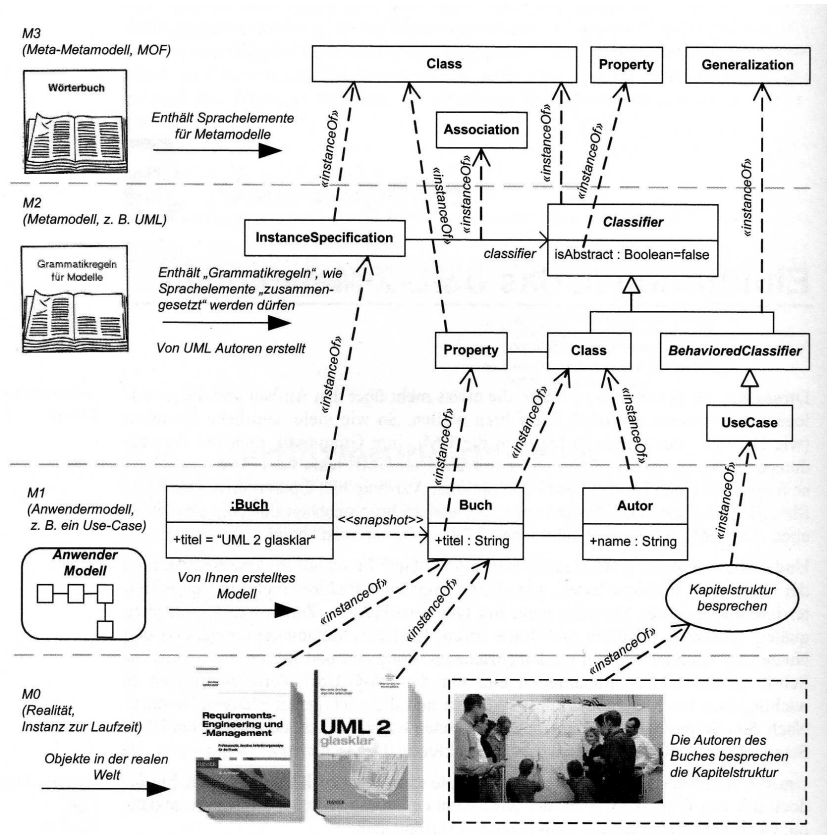

Abbildung 1: Vier Schichten Modell um die UML [RHQ+05, S. 54]

Es gibt also vor allem zwei Optionen im Eclipse-EMF-Umfeld die Struktur seiner Problemdomäne auf der Ebene M2 zu beschreiben, einmal indem man ein bestehendes Modell wie die UML an seine Bedürfnisse anpasst oder indem man sein eigenes Modell auf Ecore-Basis erstellt.

2.3 Vorgehen bei der MDD

Hat man sich festgelegt, ob man auf UML setzen will oder die Schicht unterhalb von Ecore selbst formuliert, kann man seine Problemdomäne beschreiben. Die UML kann man durch die Profile und die OCL auf seine Problemwelt anpassen [S. 9 ff.]. Dies wurde für einige Domänen bereits erfolgreich getan und es entstanden Standards der OMG, etwa BPMN für Geschäftsprozesse [BPMN, Abbildung S. 7; Fachliche Modellierung, Abbildung S. 9]. Man muss das Rad also für vieles nicht neu erfinden. Eine solche problemspezifische Sprache innerhalb einer Universalsprache, wie hier innerhalb der UML wird als interne DSL bezeichnet. Im Gegensatz dazu steht eine externe DSL, die man von Grund auf selbst definiert [Fow05].

Die Implementation und das Erstellen der notwendigen Tools für einen von der OMG definierten Standard ist dann allerdings wieder den kommerziellen oder Open-Source-Projekten vorbehalten [SVEH07, S. 72]. Genau hier soll das MDT helfen und liefert etwa mit UML2 ein Projekt, dass den UML-Standard beschreibt und mit den UML2 Tools eine Sammlung von Werkzeugen, um mit dem Standard arbeiten zu können. Die anderen Spezifikationen erweitern oftmals die UML - sie sind also als interne DSLs zur UML zu sehen.

Für den Gebrauch einer Universalsprache beim Modellieren spricht, dass man direkt starten und auf alle bewährten Werkzeuge seiner IDE zurückgreifen kann. Nachteilig ist, dass alle domänenspezifischen Fehler unentdeckt bleiben, die in der Universalsprache syntaktisch erlaubt sind. Definiert man die Sprache selbst, ist sie genau an der Problemwelt angelehnt, aber es fehlen oft die nötigen Werkzeuge zum Einsatz. Für den Bau solcher notwendiger Werkzeuge zum Modellieren, Parsen und Codegenerieren mit externen DSLs geben die Schwesterprojekte der MDT Hilfestellung. Auf diese Werkzeuge wird in dieser Arbeit nicht weiter eingegangen.

Am Ende steht dann ein domänenspezifisches Modell aus dem Programmcode für die Anwendung der Zielplattform oder ein anderes Modell erstellt werden kann.

3 MDT innerhalb des Modeling-Projects

MDT ist ein Container-Projekt. Die Unterprojekte sind „operation projects“. MDT selbst ist in der Inkubationsphase. Die Unterprojekte sind in unterschiedlichen Phasen, aktuell sind BPMN2, Information Management Metamodel (IMM), MST, Papyrus, SBVR, OCL, UML2, UML2Tools und XSD als Projekte unter dem MDT-Dach versammelt.

Das Modeling-Projekt versucht besonders bei der Erstellung eigener DSLs zu helfen, wie man aus dem Untertitel des EMPs „A Domain-Specific Language Toolkit“ erkennen kann. MDT läuft dabei etwas nebenher und sorgt dafür, dass die bestehenden Spezifikationen dabei genutzt werden können, allerdings eben auch für sich alleine verfügbar sind, etwa UML2 zum Gebrauch der UML-Spezifikation. Darüber hinaus sollen Werkzeuge zum Gebrauch der Spezifikationen geliefert werden, etwa UML2Tools, die die Arbeit mit den Standards ermöglichen. Man wird sehen, dass die Spezifikationen der OMG teilweise als interne DSL der Universalsprache UML umgesetzt sind, genauer sie erweitern oftmals die UML durch Profile [Kapitel S. 9 ff.]. Mit den eben genannten Grundlagen ist es nun möglich, die Rolle und Ideen zu verstehen, die hinter den einzelnen MDT-Unter-Projekten stecken [Ver09a].

MST hat zum Ziel den passenden Rahmen zu schaffen, um mit dem MOF-Metametamodell arbeiten zu können. Die UML-Spezifikation leitet sich von MOF ab [Abbildung S. 4]. Im Eclipse-Umfeld wurde UML aber von Ecore als Metamodell beschrieben. MST hat nun zur Aufgabe, dass das MOF-Metamodell verfügbar ist, falls dies einmal notwendig werden sollte. MST regelt auch die Serialisierung von Modellen im XMI-Format. Zukünftig soll MST eine Hilfe beim Bau von Spezifikationen sein, also insbesondere interessant für die Beteiligten der OMG [Ver09c].

UML2 ist die Implementierung des UML-Standards. UML2 liefert allerdings keine Werkzeuge, sondern nur die Implementierung des UML-Standards, Testfälle, Validierungsfälle und ein XMI-Schema, damit Modelle serialisiert und ausgetauscht werden können [Ver09d].

Wie UML2 ist auch das OCL-Projekt die Implementierung des OCL-Standards. Es werden keine Tools zum Umgang angeboten. Wie schon erwähnt, ist die OCL eine Abfragesprache für UML-Diagramme und lässt es so zu, dass man die Anforderungen an ein Diagramm deutlich genauer formulieren und auch prüfen kann. Das zugehörige OCL-Tools-Projekt ist allerdings eingestellt, der Termination Review wurde am 8. Oktober 2008 eingereicht[Ver09a].

Die Projekte BPMN2, IMM, SBVR sind interne DSLs, dabei ist UML die Hostsprache. BPMN2 implementiert den BPMN-Standard der OMG in der kommenden Version 2. Dabei ist das Ziel der BPMN die grafische Darstellung der Geschäftsprozessmodellierung. Es werden also Symbole angeboten, mit denen dann eine Folge von Tätigkeiten modelliert werden kann. Seit 2005 befindet sich diese Domäne unter dem Dach der OMG als einer der Standards. Die grafischen Elemente werden eingeteilt in Flow Objects (Knoten im Geschäftsprozessdiagramm), Connection Objects (Kanten im Diagramm), Swimlanes (Bereichsabgrenzung für Bereiche) und Artifacts (weitere Elemente). BPMN [Abbildung S. 7] spielt mit SBVR zusammen: SBVR liefert die Norm für Fachbegriffe, Geschäftsobjekte und Geschäftsregeln für BPMN [BA07, S. 3]. SBVR ist eine Implementierung eines Metamodells für Geschäftsregeln und Geschäftsvokabular, SBVR ist ebenfalls ein OMG-Standard. Dabei wird es durch die Spezifikation möglich, dass eine Transformation des Platform Independent Model (PIM) zum Platform Specific Model (PSM) gelingt. Der Wechsel vom PIM zum PSM ist ein spezieller Abschnitt des von der OMG patentierten MDA-Standards [Etz06, S. 11].

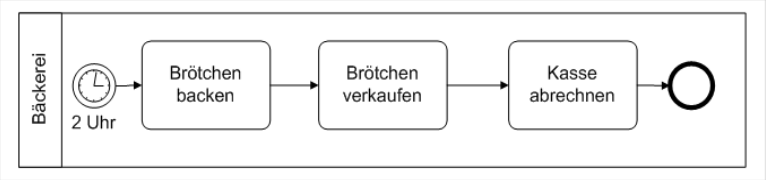

Abbildung 2: Business Process Modeling Notation Standard, aus [Ver09g]

IMM ist wie UML2, OCL, BPMN2 und SBVR ein Projekt, dass den Standard der OMG implementiert. Der OMG-Standard soll dabei den bestehenden Standard „Common Warehouse Metamodel“ (CWM) ablösen. CWM soll durch IMM abgelöst werden, damit IMM im MDA-Prozess verwendet werden kann. CWM ist nicht so formuliert, damit eine Codegeneration möglich ist und der Name spiegelt den Aufgabenbereich des Profils nicht angemessen wider, die beschriebene Domäne geht weit über Data Warehouses hinaus[Ver09f]. Dabei soll das Projekt auch die Chance nutzen, die Kooperation der OMG und der Eclipse-Welt zu intensivieren [Ver09b].

Papyrus soll ein Editor für UML und „Systems Modeling Language“ (SysML) werden. SysML ist eine Sprache für die Modellierung komplexer Systeme, an denen neben Softwarefachleuten auch Elektrotechniker und Regelungstechniker mitwirken [Wol07]. SysML erweitert die UML [Ver09j] und ist selbst auch ein OMG-Standard. SysML bietet teils andere Diagrammarten wie die UML, etwa Systemkontextdiagramme und dazu die profilüblichen eigenen Stereotypen innerhalb der Diagramme. Zusätzlich gibt es Konventionen bei den Kommentaren, um so die Aussagen zu kategorisieren [Abbildung S. 8] und so die Aussagekraft zu erhöhen. Der Praxisnutzen von SysML ist umstritten [PR08, Teil 4, Fazit]. Papyrus soll einmal selbst zu einem Containerprojekt werden und dabei viele andere Projekte unter sich bündeln, damit ein MDD-Werkzeug entsteht, welches eine Open-Source-Alternative zu den kommerziellen Spitzenwerkzeugen darstellt. Entwickelt wird Papyrus von einem Konsortium unter Federführung von LISE, einem Team einer französischen Atombehörde. Papyrus baut dabei auf UML2 auf, arbeitet in Kombination mit TOPCASE, einer Model-Based-Engineering-Platform und MOSKitt einem Editor, der die UML2 Tools erweitert und eine Kompatibilität mit verschiedenen CASE-Tools sicherstellen soll. Papyrus soll also ein sehr umfassendes Projekt im Rahmen der modellgetriebenen Entwicklung werden [Ken09].

UML2 Tools stellt die grafischen Werkzeuge zur Verfügung, dass mit dem UML-Standard im Eclipse-Umfeld gearbeitet werden kann, also UML2-Modelle wirklich erstellt und bearbeitet werden können. Es ist daher eine Konkurrenz zu Papyrus, obwohl Papyrus auch längerfristig plant, die UML2-Tools irgendwann einmal abzulösen bzw. zu vereinnahmen. UML2 Tools basiert auf GMF. Aktuell ist die Version 0.9 und unterstützt werden Strukturdiagramme (Klassendiagramm, Profildefinition, Kompositionsdiagramm, Komponentendiagramm, Deploymentdiagramm), Verhaltensdiagramme (Aktivitätsdiagramm, Zustandsdiagramm, Anwendungsfalldiagramm), Interaktionsdiagramme (Sequenzdiagramm,

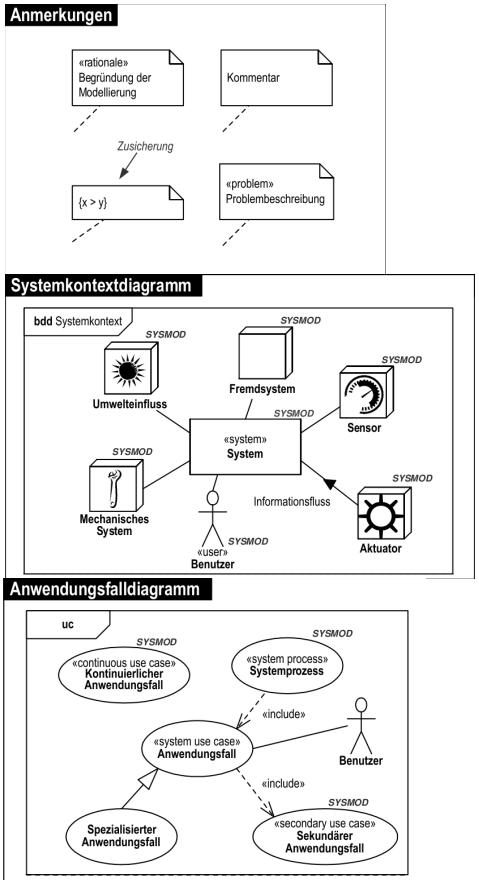

Abbildung 3: Diagramme und -elemente der SysML mit profilbezogenen Stereotypen und Kommentarkonventionen, Abbildung nach [Tim08]

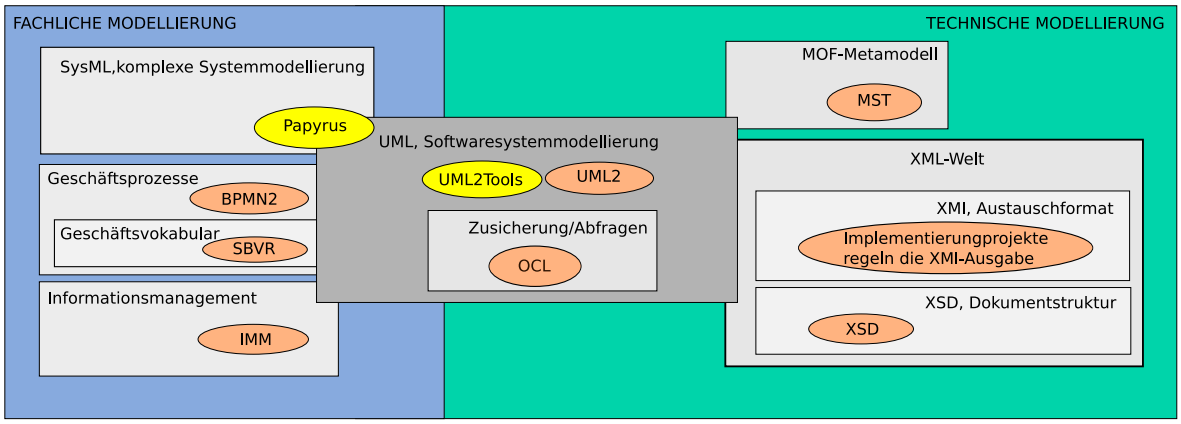

Abbildung 4: Zusammenspiel von Standards und Projekten im MDT

Timingdiagramm (noch unveröffentlicht)). Das Projekt baut natürlich auf UML2 auf[Ver09e].

XSD ist die Implementierung des OMG-Standards XML Schema Definitions (XSD). Die Möglichkeit zur Arbeit mit XSD ist sehr wichtig, da EMF-Modelle mit einer XML-basierten Sprache beschrieben werden können und XSD gibt hier die Möglichkeit die Struktur von XML-Dokumenten festzulegen, also die Regeln zu beschreiben und die dann nach den Regeln erstellten Modelle automatisch auf Regeleinhaltung zu überprüfen. XSD spielt somit überall eine Rolle, wenn mit XML-Dokumenten gearbeitet wird.

Das Zusammenspiel der MDT-Projekte soll durch die Abbildung auf S. 9 verdeutlicht werden: die UML-Profile erweitern die UML und passen diese an eine bestimmte Domäne fachlich an. MST und die XML-Standards befinden sich eher auf der Seite der technischen Modellierung. Die Gestaltungsvorschrift der XMI-Datei zum jeweiligen Modell wird vom Implementierungsprojekt festgelegt, also legt etwa UML2 für UML-Modelle fest, wie diese als XMI-Datei abzubilden sind. Die Tabelle auf S. 10 zeigt, welches Projekt zu welchem Standard gehört und gibt einen Hinweis auf den Aufgabenbereich oder die Versionsnummer des Projekts.

4 Profile und OCL bei internen DSLs auf UML-Basis

Will man Software modellgetrieben, aber ohne eigens formulierte Sprache entwickeln, wird man gewöhnlich auf UML setzen, sofern nichts gegen einen solchen Einsatz spricht. Bei Bedarf hält man Ausschau, ob ergänzende Standards für die eigene Domäne bereits existieren. Bei diesen internen DSLs spielen die OCL und UML-Profile als Erweiterungsmechanismen von UML eine wichtige Rolle.

OCL erweitert die UML und hat dabei keinerlei Seiteneffekte. Das bedeutet, dass das UML-Modell durch die OCL nicht geändert, sondern lediglich überwacht wird. Durch

Projekt	Standard	Bemerkung
MDT	-	Version 1.1; Containerprojekt
BPMN2	BPMN	Version 0.7; Geschäftsprozessmodellierung
IMM	IMM	Metamodell-Standard für Informationsmanagement
MST	MOF	EMF-Aufsatz für Spezifikationsersteller
OCL	OCL	Version 3; 5. Release
Papyrus	-	Version 0.7; Ziel Integrated Modeling Environment
SBVR	SBVR	Metamodel-Standard für Geschäftsdomäne/ -vokabular
UML2	UML	Version 3.1; Metamodel UML-Implementierung für EMF
UML2 Tools	-	Version 0.9; Werkzeuge für UML-Einsatz
XSD	XSD	Version 2.6; XML-Schema-Datei-Definition

Tabelle 1: Übersicht von MDT und seiner Unterprojekte

diese Überwachung ist es möglich die Anforderungen an das Modell genauer festzulegen. Auch innerhalb der UML-Beschreibung taucht die OCL auf, etwa wird so festgehalten, dass Generalisierungsbeziehungen gerichtet sein müssen und nicht zyklisch sein dürfen [Ver07, S. 81]. Gronback und Merks stellen zur Rolle der OCL innerhalb des EMP fest: „It's Everywherere (time to learn it!)“ [GM08, S. 6]. Die OCL-Ausdrücke im UML-Modell haben folgende Spezifizierungsabsichten [DM05, S. 56]:

- Invariante Bedingungen für Klassen und Typen
- Vor- und Nachbedingungen für Methoden
- Abfrageausdrücke
- Ableitungsregeln für berechenbare Attribute

Festgehalten werden die OCL-Ausdrücke entweder als Kommentar [Abbildung S. 13] im UML-Modell oder rein textuell an anderer Stelle. OCL-Ausdrücke sind immer an einen Kontext gebunden [Liste S. 10]. Durch die OCL wird es also möglich, das „Design By Contract“ [3] im Modell festzuhalten. Durch die Formalisierung der OCL ist es eben auch möglich, dass die OCL-Anweisungen maschinell in Code umgesetzt bzw. ausgewertet werden können. Diese Automatisation ist für die MDD ausschlaggebend. Die OCL orientiert sich an der Syntax von Smalltalk, das heißt, dass das Gleichheitszeichen für einen Vergleich und nicht für eine Zuweisung steht! Folgende Abbildung auf S. 11 und die zugehörige Liste auf S. 10 zeigen ein UML-Basisdiagramm, eine ans Modell gestellte Anforderung und den zugehörigen OCL-Ausdruck.

- Bedingung: Das Alter einer Person ist nicht negativ.
  OCL-Constraint: **context** Person **inv**: self.Alter >= 0

[3] Entwurf gemäß Vertrag, festgelegt von Bertrand Meyer: durch die Festlegung von Invarianten, Vor- und Nachbedingungen der interagierenden Softwarebausteinen soll ein reibungsloser Programmablauf sichergestellt werden.

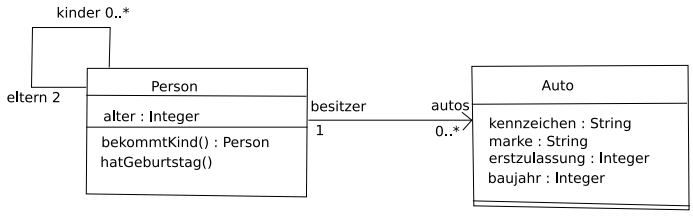

Abbildung 5: UML-Basis-Diagramm zur Veranschaulichung von Constraints, nach [Ver09i]

- Bedingung: Eine Person ist jünger als ihre Eltern.
  OCL-Constraint: **context** Person **inv**: self.Eltern->forAll(e|e.Alter>self.Alter)
- Bedingung: Nur eine erwachsene Person darf ein Auto besitzen.
  OCL-Constraint: **context** Person **inv**: self.Alter<18 **implies** autos->size()=0

Die OCL überwacht also, ob ein Modell eingehalten wird, die Constraints wirken dabei als Einschränkung. Jetzt möchte man vielleicht die Universalsprache UML wirklich erweitern. Hier helfen die UML-Profile. Ein Profil ist ein „Stereotypisiertes Paket, das Modellelemente der UML enthält, die für einen bestimmten Einsatzzweck angepasst werden ...“ [Bal05, S. 542]. Mit der OCL und den UML-Profilen ist es dann möglich eine interne DSL zu bilden - dabei muss man auf die Annehmlichkeiten des Arbeitens in der Universalsprachen-Umgebung sogar nicht einmal verzichten. Neben dem Einsatzzweck ein Profil für eine bestimmte Domäne zu definieren, kann man auch eines erstellen, dass die UML auf eine bestimmte Plattform anpasst. Es ist also denkbar ein Profil Java oder EJB anzulegen [siehe Abbildung S. 13].

Stereotype stehen in französischen Anführungszeichen, den Guillements, über dem Bezeichner der Klasse. Für sich genommen, weisen Stereotype erst einmal nur auf eine besondere Charakteristik des Elements hin. Ein Stereotyp muss erst definiert werden, bevor man diesen einsetzen kann. Dies geschieht, indem man das Wort „stereotype“ in Guillements oberhalb des Element-Bezeichners notiert und als Element-Bezeichner den Namen des zu definierenden Stereotyps schreibt. Darunter folgen die Attribute des Stereotyps. Wird dieser Stereotyp-Bezeichner dann an anderer Stelle in Guillements gesetzt, ist klar, mit welcher besonderen Charakteristik dieses Element verknüpft ist. So wendet man Stereotype dann an. Bei der ABEweb-Methode zur Entwicklung von Webapplikationen werden die Aktivitätsdiagramme der UML zu interaktionsorientierten Aktivitätsdiagrammen erweitert. In diesen Diagrammen kommen Aktionen vor, die mit den Stereotypen „SA“ oder „AA“ versehen sind. So ist klar, ob es sich um eine Akteuraktion oder eine Systemaktion handelt und diese Unterscheidung ist später für die Transformation sehr wichtig[LS04, S. 255 ff.]. Die Abbildung auf S. 12 zeigt den eben erwähnten Einsatz der Stereotypen ≪SA≫und ≪AA≫.

Bündelt man nun ein Paket mit Stereotypen und gibt diesem Paket selbst den Stereotyp

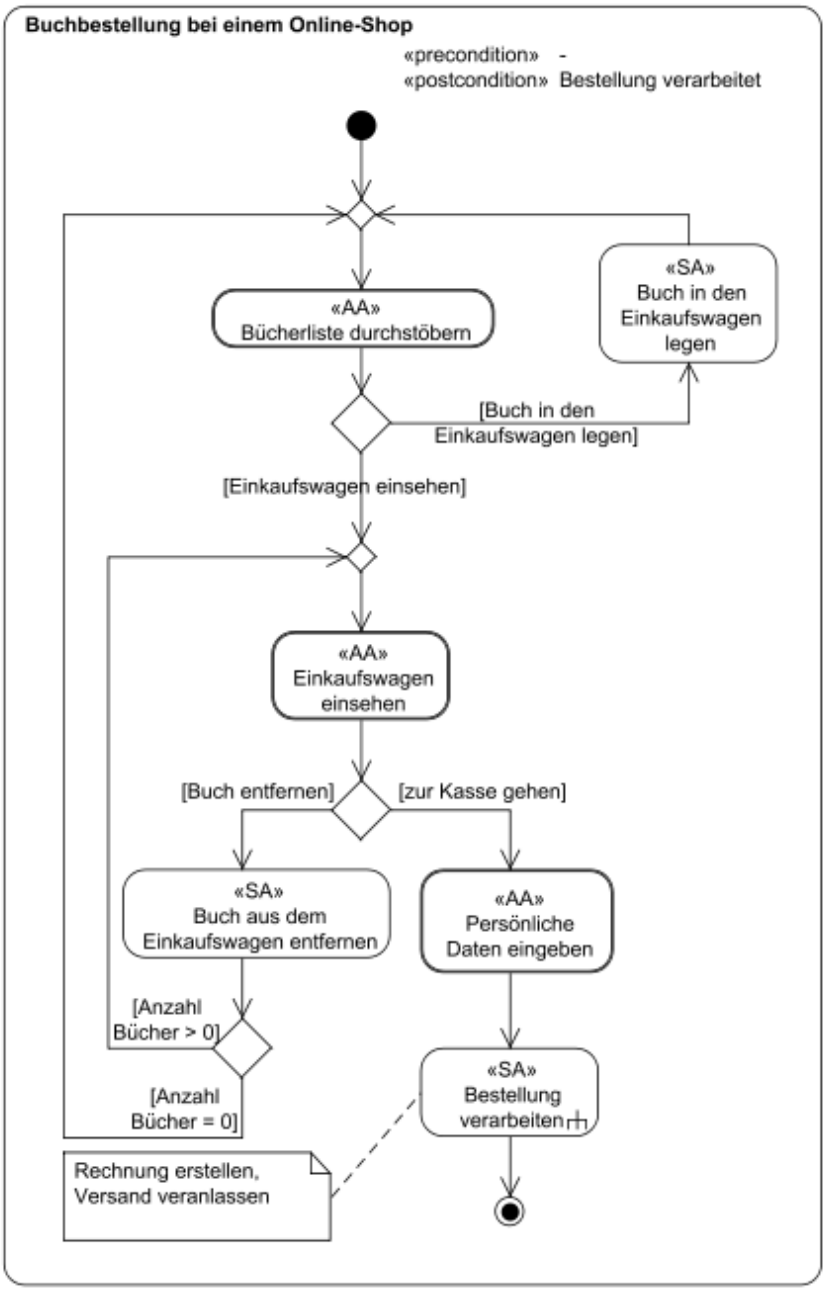

Abbildung 6: Stereotypisiertes interaktionsorientiertes Aktivitätsdiagramm zum Anwendungsfall „Buchbestellung bei einem Online-Shop“ , Abbildung aus [LS04, S. 250]

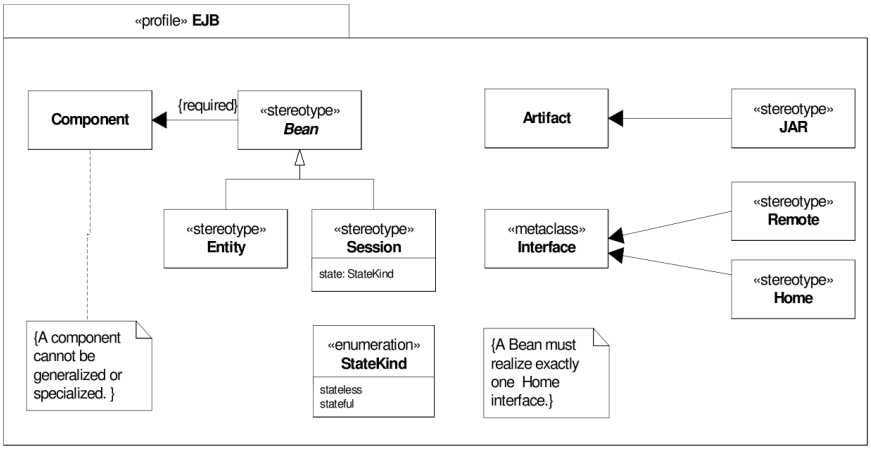

Abbildung 7: Definition eines UML-Profils, hier für EJB, Abbildung aus [Ver07, S. 188]

„profile“ , dann erhält man ein UML-Profil, wie bereits erwähnt, handelt es sich dabei um ein stereotypisiertes Paket. Die Abbildung auf S. 13 zeigt die Definition eines UML-Profils. Der Paketname ist durch den Stereotyp „profile“ ausgezeichnet, im Profil sind nur Stereotypen, die ein Element der Metaebene erweitern. Zusätzlich befinden sich in den Kommentaren die Randbedingungen in Form von OCL-Ausdrücken.

Das Modell seines modellierten Systems kann man ebenfalls in ein Paket stecken und dieses Paket kann dann ein oder mehrere UML-Profile einbinden - so sind die in den eingebundenen Profilen definierten Stereotypen auch im Paket des modellierten Systems sichtbar. Die Abbildung auf S. 14 zeigt diese „apply“-Beziehung zur Profileinbindung. Die UML wurde so durch ein oder mehrere Profile erweitert und ist gegebenenfalls so zu einer internen DSL geworden. Die Modelle des Pakets Webshopping können nun auf alle Stereotypen zugreifen, die in den Profilen Java und EJB definiert wurden.

Beim Erstellen von Profilen muss einiges bedacht werden, etwa muss gesichert sein, dass die Profile nicht mit der UML im Widerspruch stehen. Der Austausch von Modellen und Profilen kann mittels des Standards XMI erfolgen. Die OMG hat bereits einige Standardprofile geschaffen, wie man im Kapitel S. 5 ff. nachlesen kann.

5 Diskussion

Theoretisch gibt es einige Vorteile der MDD. Zu allererst ist ein Domänenexperte besser einbindbar: er ist ein Fachmann im Problemfeld und findet sich daher im Umfeld einer domänenspezifischen Sprache besser zurecht als im Umfeld einer 3-GL-Programmiersprache, die er noch nicht kennt. Dazu altert das DSL-System mit der Domäne und nicht mit der eingesetzten Technik. Darüber hinaus kann der Code automatisch erzeugt werden - es

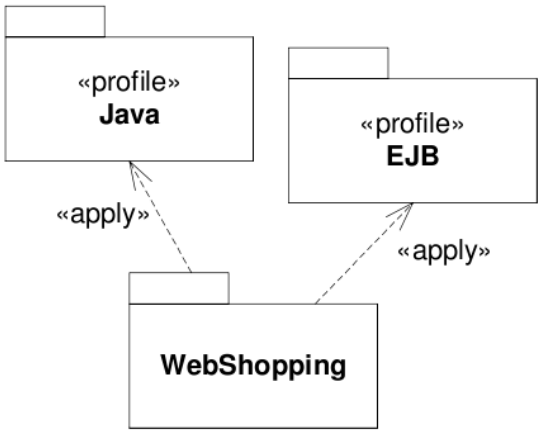

Abbildung 8: Anwendung eines UML-Profils auf ein Modell, Abbildung aus [Ver07, S. 191]

werden also die Fehler verhindert, die beim Implementieren anfallen und viel Zeit kosten.

Die Idee der automatischen Codegeneration gibt es schon länger - die CASE-Werkzeuge in den 90er Jahren wollten dies ebenfalls leisten. Die Abgrenzung zwischen CASE- und MDD-Werkzeugen besteht nun darin, dass bei den CASE-Werkzeugen eine Vollautomatisierung das Ziel ist: aus einer Systembeschreibung soll eine fertige Architektur erzeugt werden. Die MDD-Ansätze wollen nur einen sinnvollen Anteil an Automatisierung bereitstellen, sich dabei aber die nötige Flexibilität erhalten.

Eine Trennung von fachlichen und technischen Anteilen führt zu einer höheren Abstraktion des Systems und die Komplexität wird so handhabbarer, einer der grundlegenden Wünsche der Softwareentwicklung, wie in der Einleitung auf S. 1 bereits angesprochen wurde. Dazu soll durch Standardisierung eine verbesserte Interoperabilität erreicht werden [Ver09h].

Die Grundvoraussetzung für die MDD, also das domänenspezifische Modell, muss von einem Entwickler allerdings erst einmal erstellt werden können. Der Entwickler hat zu entscheiden, ob er einen Standard einsetzt, oder eine interne DSL bevorzugt, oder eine externe DSL erzeugt. Ein erfahrener Entwickler wird den anfallenden Aufwand für die drei Varianten einschätzen können. Fowler, der die Begriffe interne und externe DSL prägte, schätzt die Wahrscheinlichkeit für den Eintritt der theoretischen Vorteile wie folgt ein: „Only time will tell if these advantages will actually be realized“ [Fow05, Conclusion].

Die Standardisierung und intensivierte Zusammenarbeit zwischen OMG und der Eclipse-Welt wird den Aufwand für das Erstellen der Generatoren und Tools für die domänenspezifischen Sprachen weiter minimieren und so dieses Feld für Einsteiger immer zugänglicher

machen. Dazu wird durch die Verfügbarkeit der Projekte unter dem MDT-Dach es eben auch möglich nur die bereits bestehenden Standards isoliert einzusetzen und die bereits vorhandenen Werkzeuge zu nutzen. Falls man von externen Anwendungen kommt, kann man über Standards wie XMI trotzdem die Modelle für die Eclipse-Welt verfügbar machen; Tools wie Papyrus sollen diese Möglichkeiten weiter optimieren und vorantreiben.

Das MDT hilft also beim Umgang mit Standards und diese helfen im MDD-Prozess mittels interner DSLs. Darüber hinaus ist der bloße Gebrauch der Standards und dazugehöriger Werkzeuge für domänenspezifische Modelle vielleicht nicht sehr hilfreich, aber für den Entwicklungsalltag trotzdem eine enorme Erleichterung. Ebenfalls birgt es viel Potential, wenn die OMG und Eclipse-Welt deren Zusammenarbeit intensivieren, da so Standards und zugehörige Implementierungen vielleicht schneller entstehen und ggf. Firmen sich entschließen die Ressourcen zu bündeln: Die Implementierung ist so zeitnah vorhanden und auch für die Open-Source-Welt verfügbar.

6 Abkürzungsverzeichnis

BPMN Business Process Modeling Notation;
CASE Computer Aided Software Engineering;
CWM Common Warehouse Metamodel;
DSL Domain specific language;
EMF Eclipse Modeling Framework;
EMOF Essential Meta Object Facility;
EMP Eclipse Modeling Project;
GEF Graphical Editing Framework;
GMF Graphical Modeling Framework;
IDE Integrated Development Environment;
IMM Information Management Metamodel;
MDA Model Driven Architecture;
MDD Model Driven Development;
MDSD Model Driven Software Development;
MDT Model Development Tools;
MOF Meta Object Facility;
MST Metamodel Specification Tools;
OCL Object Constraint Language;
OMG Object Management Group;
PIM Platform Independent Model;
PSM Platform Specific Model;
SVBR Semantics of Business Vocabulary and Business Rules;
SysML Systems Modeling Language;
UML Unified Modeling Language;
XMI Extensible Markup Language Metadata Interchange;
XSD Extensible Markup Language Schema Definition;

Literatur

[BA07] Bartonitz, Michael ; Ammon, Rainer von: BPMN Relevante Standards. In: *Enterprise Architecture und BPM* (2007), S. 1–5. – Onlineausgabe

[Bal05] Balzert, Heide: *Lehrbuch der Objektmodellierung, Analyse und Entwurf mit der UML2*. Muenchen : Elsevier GmbH, 2005. – 2. Auflage

[DM05] Demelt, Achim ; Mitrik, Dieter: OCL in der Praxis. In: *JavaSpektrum* (2005), Nr. 01, S. 56–59. – Onlineausgabe

[Etz06] Etzel, Marcel: *Modellgetriebene Softwareentwicklung*. Website, Seminararbeit, Johann Wolfgang Goethe-Universitaet, Frankfurt, 2006. – www.is-frankfurt.de/uploads/down568.pdf

[Fow05] Fowler, Martin: *Language Workbenches, The Killer-App for Domain Specific Languages?* Website, Juni 2005. – http://martinfowler.com/articles/languageWorkbench.html [zuletzt besucht am 02.12.2009]

[GM08] Gronback, Richard ; Merks, Ed: *Eclipse Modeling Project*. Webinar, Maerz 2008

[Ken09] Kenn, Hussey: *Papyrus, Advent of an Open Source IME at Eclipse*. Website, 2009. – http://www.slideshare.net/kenn.hussey/papyrus-advent-of-an-open-source-ime-at-eclipse [zuletzt abgerufen am 05.12.2009]

[LS04] Lorenz, Alexander ; Six, Hans-Werner: *Software Engineering 2, Methodische Entwicklung von Webapplikationen*. Hagen : FernUniversitaet Hagen, 2004. – Kurs 01794, Wintersemester 2009/2010

[PR08] Pflug, Carsten ; Rupp, Chris: *Haelt die SysML was sie verspricht?* Website, 2008. – http://it-republik.de/jaxenter/artikel/Haelt-die-SysML-was-sie-verspricht-1531.html [zuletzt besucht 21.12.2009]

[RHQ+05] Rupp, Chris ; Hahn, Juergen ; Queins, Stefan ; Jeckle, Mario ; Zengler, Barbara: *UML 2 glasklar, Praxiswissen fuer UML-Modellierung und -Zertifizierung*. Muenchen : Carl Hanser Verlag, 2005. – 2. Auflage

[SVEH07] Stahl, Thomas ; Voelter, Markus ; Efftinge, Sven ; Haase, Arno: *Modellgetriebene Softwareentwicklung, Techniken Engineering Management*. Heidelberg : dpunkt, 2007. – 2. Auflage

[SW02] Six, Hans-Werner ; Winter, Mario: *Software-Engineering 1, Methodische Entwicklung objektorientierter Desktop-Applikationen*. Hagen : FernUniversitaet Hagen, 2002. – Kurs 1793, Sommersemester 2009

[Tim08] Tim, Weilkins: *SysML-Notationsuebersicht*. Website, 2008. – http://www.oose.de/downloads/sysml_notationsuebersicht.pdf [zuletzt abgerufen am 21.12.2009]

[Ver07] Verfasser ohne: *Unified Modeling Language, Infrastructure*. Website, 2007. – Version 2.1.1

[Ver09a] Verfasser ohne: *Eclipse, MDT*. Website, 2009. – http://www.eclipse.org/modeling/mdt/ [zuletzt abgerufen am 04.12.2009]

[Ver09b] Verfasser ohne: *Eclipsepedia, MDT-IMM*. Website, 2009. – http://wiki.eclipse.org/MDT/IMM [zuletzt abgerufen am 05.12.2009]

[Ver09c] Verfasser ohne: *Eclipsepedia, MDT-MST*. Website, 2009. – http://wiki.eclipse.org/MDT/MST [zuletzt abgerufen am 05.12.2009]

[Ver09d] Verfasser ohne: *Eclipsepedia, MDT-UML2*. Website, 2009. – http://wiki.eclipse.org/MDT/UML2 [zuletzt abgerufen am 05.12.2009]

[Ver09e] Verfasser ohne: *Eclipsepedia, MDT-UML2Tools*. Website, 2009. – http://wiki.eclipse.org/MDT-UML2Tools [zuletzt abgerufen am 05.12.2009]

[Ver09f] Verfasser ohne: *OMG-Wiki, The IMM story - some background*. Website, 2009. – http://www.omgwiki.org/imm/doku.php [zuletzt abgerufen am 05.12.2009]

[Ver09g] Verfasser ohne: *Wikipedia, Business Process Modeling Notation*. Website, 2009. – http://de.wikipedia.org/wiki/BPMN [zuletzt abgerufen am 02.12.2009]

[Ver09h] Verfasser ohne: *Wikipedia, Model Driven Architecture*. Website, 2009. – http://de.wikipedia.org/wiki/Model_Driven_Architecture [zuletzt abgerufen am 22.12.2009]

[Ver09i] Verfasser ohne: *Wikipedia, Object Constraint Language*. Website, 2009. – http://de.wikipedia.org/wiki/Object_Constraint_Language [zuletzt abgerufen am 03.12.2009]

[Ver09j] Verfasser ohne: *Wikipedia, Systems Modeling Language*. Website, 2009. – http://de.wikipedia.org/wiki/Systems_Modeling_Language [zuletzt abgerufen am 02.12.2009]

[Wol07] Wolff, Viviane: *SysML und UML als Werkzeuge zur Systemmodellierung eingebetteter Systeme*. Website, 2007. – http://www.fh-fulda.de/fileadmin/Fachbereich_ET/Schaukasten/Schaukasten_2007/ET-Seminar/Dateien/ET-Seminar2007_Wolff_HS_Fulda.pdf [abgerufen am 21.12.2009]

Hiermit versichere ich, dass die vorliegende Arbeit selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Sulzbach-Rosenberg, 8.12.2009