Technische Hochschule Köln
TH Köln – University of Applied Sciences
Campus Gummersbach
Fakultät für Informatik und Ingenieurwissenschaften

Fachhochschule Dortmund
University of Applied Sciences and Arts
Fachbereich Informatik

Verbundstudiengang Wirtschaftsinformatik

Abschlussarbeit

zur Erlangung

des Bachelorgrades

Bachelor of Science

in der Fachrichtung Informatik

„Modellgetriebene O/R-Mapper: Überblick und Vergleich“

Erstprüfer:	Prof. Dr. Heide Faeskorn-Woyke
Zweitprüfer:	Prof. Dr. Birgit Bertelsmeier
vorgelegt am:	20.06.2016
von cand.	Christian Herrmann
aus	Bollinghausen 3 42929 Wermelskirchen
Tel.:	02196/8822737
Email:	christianblitz@msn.com
Matr.-Nr.:	11082914

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungs- u. Symbolverzeichnis

ACID	Eigenschaften einer Transaktion (atomar, konsistent, isoliert und dauerhaft)
CRUD	Create Read Update Delete
DDL	Data Definition Language
EAL	Eco Action Language
EMF	Eclipse Modeling Framework (s. Kapitel 3.2.1)
EMFT	Eclipse Modeling Framework Technologies (s. Kapitel 3.2.1)
EPL	Eclipse Public License
ERM	ER-Modell, Entity Relationship Model
GUI	Graphical User Interface (Benutzeroberfläche)
IDE	Integrated Development Environment (integrierte Entwicklungsumgebung)
IoC	Inversion of Control (Entwurfsmuster)
JPA	Java Persistence API (standardisierter Persistenzzugriff in Java u. a. mittels O/R-Mapping)
JPQL	Java Persistence Query Language
LINQ	Language Integrated Query
M2M	Modell-zu-Modell-Transformation (s. Kapitel 2.1.3)
M2T	Modell-zu-Text-Transformation (s. Kapitel 2.1.3)
MMT	s. M2M
MDA	Model Driven Architecture (s. Kapitel 2.1.2)
MDSD	Model Driven Software Development (s. Kapitel 2.1.1)
MOF	Meta Object Facility
OCL	Object Query Language
OMG	Object Management Group, ein Industriekonsortium
ORM, O/R-Mapping	Object Relational Mapping (objektrelationale Abbildung, s. Kapitel 2.2.2)
O/R-Mapper	Object Relational Mapper (Softwarekomponente, die sich um die objektrelationale Abbildung kümmert, s. Kapitel 2.2.3)
PIM	Platform Independent Model

POJO	Plain Old Java Object: einfache Java Objekte ohne Abhängigkeiten
PSM	Platform Specific Model
SaaS	Software as a Service
SQL	Structured Query Language
UML	Unified Modeling Language (standardisierte Modellierungssprache der OMG)
XML	Extensible Markup Language
XMI	XML Metadata Interchange

1 Das Besondere an modellgetriebenen O/R-Mappern

Zu den Themen der objektrelationalen Abbildung (engl. object-relational mapping, ORM) und der praktischen Umsetzung dieser Abbildung mit Hilfe von O/R-Mappern wie z. B. Hibernate gibt es ausreichend Literatur. Auf die Besonderheiten der modellgetriebenen Softwareentwicklung wird dabei aber eher selten eingegangen. Häufig erfolgt das Mapping von Objekten zu den relationalen Datenbanktabellen über Annotationen im Quellcode oder in gesonderten XML-Dateien, anstatt es zentral und standardisiert in einem Modell anzugeben. Es gibt nur wenige O/R-Mapper, die sich auf ein Modell als Kern und Ausgangspunkt der Anwendungslogik und Persistenzhaltung spezialisiert haben. Häufig geht bei diesen modellgetrieben O/R-Mappern der Umfang weit über das reine O/R-Mapping hinaus, weshalb in dieser Arbeit auch der Begriff Framework genutzt wird. Da sich damit die Möglichkeiten über denen von normalen O/R-Mapper bewegen, lohnt sich eine genauere Beschäftigung mit diesen Frameworks in besonderem Maße.

Diese Arbeit besteht im Wesentlichen aus zwei Teilen, die jeweils ein übergeordnetes Ziel verfolgen. Im ersten Teil sollen die Besonderheiten des modellgetriebenen Ansatzes im Zusammenhang mit der objektrelationalen Abbildung aufgezeigt und erläutert werden. Dazu gehört auch eine Diskussion der Vor- und Nachteile gegenüber normalen O/R-Mappern. Der Leser soll dann entscheiden können, ob und in welchen Situationen der modellgetriebene Ansatz sinnvoll ist. Dieser Teil ist hauptsächlich als Literaturarbeit zu verstehen und soll dem Überblick über das Thema und der Vorbereitung auf den zweiten Teil dienen. Im zweiten Teil erfolgt dann ein konstruktiver Vergleich von modellgetriebenen O/R-Mapping-Frameworks, mit dem Ziel, die Stärken und Schwächen sowie Besonderheiten der jeweiligen Frameworks herauszufinden. Damit soll der Leser in die Lage versetzt werden, sich abhängig vom Anwendungsfall für ein bestimmtes modellgetriebenes O/R-Mapping-Framework zu entscheiden.

Diese Arbeit ist keine Arbeit über die objektrelationale Abbildung und O/R-Mapper im Allgemeinen, auch wenn diese zum besseren Verständnis nochmals erläutert werden. Der Fokus liegt auf dem modellgetriebenen Ansatz und den entsprechenden modellgetriebenen O/R-Mapping-Frameworks. Andere herkömmliche O/R-Mapper sind bewusst aus dem Vergleich ausgenommen, da bei diesen, wie noch erörtert wird, andere Anforderungen relevant sind und ebenso die Einsatzszenarien unterschiedlich

sein können. Weiterhin soll es um die Persistierung von Geschäftsdaten gehen, also um die Speicherung wichtiger Daten für den geschäftlichen Betrieb, für die es gewisse Anforderungen hinsichtlich Konsistenz und Sicherheit gibt. Datenintensive Web 2.0-Anwendungen sind daher ausgenommen.

Das Kapitel 2 umfasst den gesamten ersten Teil der Arbeit. Um einen Einstieg in das Thema zu finden, werden zunächst einige Grundlagen behandelt und Definitionen aufgestellt zu den Themen: modellgetriebene Softwareentwicklung (Kapitel 2.1) und objektrelationale Abbildung (Kapitel 2.2). Das Ergebnis ist die Zusammenführung der beiden Themen sowie die Definition des Begriffs „modellgetriebener O/R-Mapper“ (Kapitel 2.3). In Kapitel 3 beginnt der zweite Teil der Arbeit mit der Konzeption des Vergleichs, bevor dieser in Kapitel 4 mit ausgewählten modellgetriebenen O/R-Mapping-Frameworks durchgeführt wird. Die Arbeit schließt mit einer Zusammenfassung und Bewertung der Ergebnisse (Kapitel 5) sowie einem Fazit und Ausblick (Kapitel 6) ab.

2 Modellgetriebener Ansatz und O/R-Mapper im Licht wissenschaftlicher Erkenntnisse

Dieses Kapitel soll zum einen der Heranführung an das Thema dienen und Definitionen aufstellen, die für diese Arbeit benötigt werden, um für eine einheitliche Begriffsbasis zu sorgen. Zum anderen soll es einen Überblick über die Themen modellgetriebene Softwareentwicklung und objektrelationale Abbildung bzw. O/R-Mapper liefern (Kapitel 2.1 bzw. 2.2) und schließlich die beiden Themen zusammenführen und den Begriff der modellgetriebenen O/R-Mapper im Detail erläutern (Kapitel 2.3).

2.1 Modellgetriebene Softwareentwicklung und der Wunsch nach Automatisierung in der Softwareentwicklung

Bereits in den 1970er Jahren kam in der Softwareentwicklung der Wunsch auf, Teile eines Programms automatisiert aus seiner Spezifikation heraus generieren zu können.[1] Erste Ansätze wurden zu dieser Zeit bereits entwickelt und eingesetzt. Dabei waren Modelle die zentralen Dokumente in der Spezifikation. Sobald diese formalisiert und in einer für die Maschine lesbaren Form vorlagen, konnte aus diesen Modellen Quellcode generiert werden. Zur Beschreibung von Modellen werden Metamodelle verwendet. Jacobson[2] nennt sogar das Jahr 1968, in dem er bereits eine formalisierte Modellierungssprache entwickelt und angewendet hatte. Diese Zeit war der Beginn der modellgetriebenen Softwareentwicklung (engl. Model Driven Software Development, MDSD).

2.1.1 Model Driven Software Development

„Die modellgetriebene Softwareentwicklung befasst sich mit der Automatisierung in der Softwareherstellung.“[3] Das Ziel ist dabei, dass möglichst viele Artefakte eines Softwaresystems aus dem Modell heraus generiert werden können. Unter einem Artefakt versteht man vereinfacht das „Ergebnis einer Tätigkeit im Rahmen der Softwareentwicklung“[4]. Neben dem reinen Quellcode können auch andere, nicht

---

[1] Vgl. Ludewig und Lichter, Software Engineering 2010, S. 340
[2] Vgl. Mellor und Balcer, Executable UML: a foundation for model-driven architecture 2002, Vorwort von Ivar Jacobson
[3] Flores Beltran, et al., Modellgetriebene Softwareentwicklung 2007, S. 11
[4] Petrasch und Meimberg, Model Driven Architecture 2006

ausführbare Dateien sowie Tests und Dokumentationen generiert werden, wie in der folgenden Übersicht beispielhaft dargestellt wird:[5]

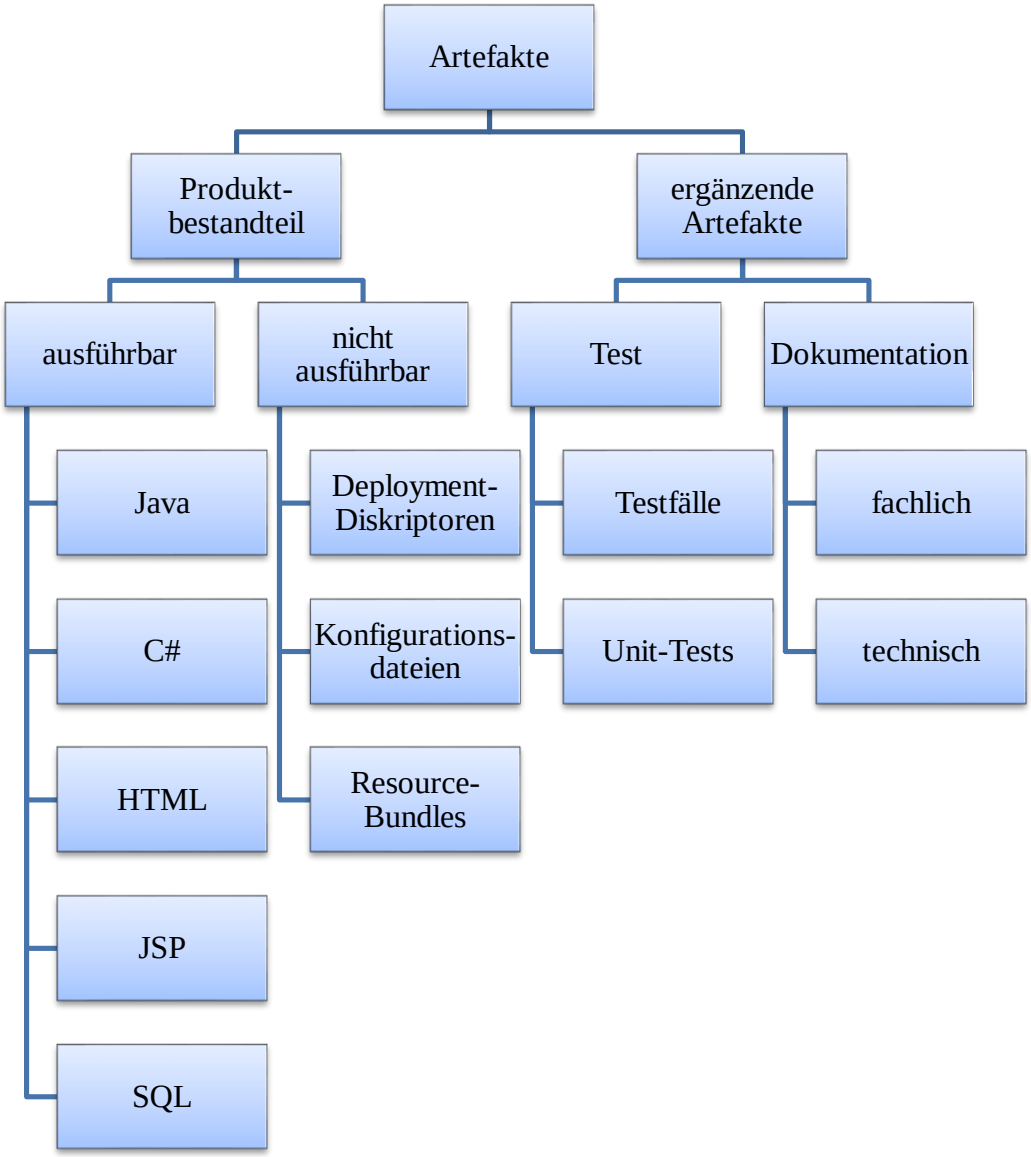


Abbildung 1: Generierbare Artefakte

Modelle bzw. Metamodelle bilden im MDSD also den zentralen Ausgangspunkt. Modelle sind dabei nicht nur ein notwendiges Übel, die ein Entwickler oder Softwarearchitekt zur Beschreibung der Anwendung oder zur Dokumentation für den Kunden erstellen muss, sondern die Modelle liefern durch die Generierung von Artefakten nebenbei einen erheblichen Mehrwert. Ohne MDSD wäre die Erzeugung

[5] Vgl. Flores Beltran, et al., Modellgetriebene Softwareentwicklung 2007, S. 11f

der Artefakte eine manuelle Tätigkeit – weder prüf- noch reproduzierbar, dafür aber fehleranfällig.[6]

Die Vorteile einer Automatisierung liegen nach Beltran et al.[7] damit in erster Linie in der gesteigerten Produktivität und Qualität bei der Softwareentwicklung. Darüber hinaus wird die Einhaltung der Phasen in der Softwareentwicklung erleichtert, da die Anpassung der Modelle bzw. auch die Erstellung selbst nicht übergangen werden kann. Außerdem kann das Vorgehen den schnelleren Wechsel auf neue Technologien und Plattformen erlauben. Dies erfolgt meist durch die Erstellung eines plattformunabhängigen Modells, welches dann entweder in plattformabhängige Modelle umgewandelt werden kann oder aus welchem plattformabhängige Artefakte generiert werden können.

2.1.2 Model Driven Architecture

Bei der modellgetriebenen Architektur (engl. Model Driven Architecture, MDA) handelt es sich um einen konkreten Ansatz der Object Management Group (OMG) zur modellgetriebenen Softwareentwicklung – mit dem Ziel, die Geschäfts- und Anwendungslogik hersteller- und plattformunabhängig zu beschreiben.[8] Seit der Gründung der OMG 1990 wurden eine Reihe von Spezifikationen veröffentlicht, die der Integration von hersteller- und plattformunabhängigen Softwaresystemen dienen sollen.[9] Um die Beziehungen und den Austausch zwischen verschiedenen Standards zu ermöglichen, wurde zunächst die Object Management Architecture (OMA) entwickelt, aus welcher dann später der MDA-Ansatz entstanden ist.[10]

Die Modellierung und damit die Unified Modeling Language (UML) stehen dabei im Mittelpunkt, obwohl grundsätzlich auch andere Modellierungssprachen unterstützt werden. Möglich wird dies, indem zusätzlich von der Ebene des Metamodells abstrahiert wird. Mithilfe der Meta Object Facility (MOF) lassen sich so die Metamodelle beschreiben, was für den Austausch von Daten bzw. Metadaten mittels

---

[6] Vgl. Flores Beltran, et al., Modellgetriebene Softwareentwicklung 2007, S. 12
[7] Vgl. dazu im Folgenden Flores Beltran, et al., Modellgetriebene Softwareentwicklung 2007, S. 23f
[8] Vgl. Flores Beltran, et al., Modellgetriebene Softwareentwicklung 2007, S. 14
[9] Vgl. Grose, Doney und Brodsky, Mastering XMI 2002, S. 328
[10] Vgl. Grose, Doney und Brodsky, Mastering XMI 2002, S. 328

des ebenfalls für die MDA standardisierten XML Metadata Interchange (XMI) wichtig ist.[11]

Zu den Vorzügen der Modellierung beim Softwaredesign zählten Grose et al.[12] in diesem Zusammenhang den einfachen Austausch der Modelle und das leichte Verständnis von Modellen durch die graphische Notation und die abstrakte Darstellung des Systems. Dies ist nicht nur für die Entwickler selbst nützlich, sondern ebenso für die Stakeholder. Schließlich sorgt ein abstraktes Modell auch dafür, den Überblick über das große Ganze zu wahren.

Nach dem MDA Guide der OMG[13] steht auch bei der MDA die automatisierte Generierung von Artefakten und Quellcode aus dem Modell heraus (ganz oder teilweise) im Vordergrund, um bei der Umsetzung des Softwaredesigns sowie bei Änderungen und Wartung Zeit und Geld zu sparen. Im Detail gibt es mehrere Modelle auf unterschiedlichen Abstraktionsniveaus, die jeweils mittels Transformationen in ein Modell mit geringerem Abstraktionsniveau überführt werden können. Dies kann so weit gehen, dass schließlich ein komplettes ausführbares System entsteht. Es wird im Wesentlichen zwischen den folgenden Kategorien für die Abstraktion unterschieden, beginnend mit der höchsten Abstraktion:

- Geschäfts- oder Domänenmodell (ehem. Computation Independent Model, CIM): Ein Modell realer Dinge, vollkommen unabhängig von der technischen Umsetzung
- Logisches Systemmodell: beschreibt, wie die Komponenten eines Systems interagieren
- Implementierungsmodell: beschreibt, wie ein konkretes System in einer bestimmten Technologie und Plattform implementiert ist
    - Platform Independent Model (PIM): Bei diesem Modell wird nur die technische Umsetzung spezifiziert, unabhängig von der verwendeten Plattform

[11] Vgl. Liddle, Model-Driven Software Development 2010, S. 16
[12] Vgl. Grose, Doney und Brodsky, Mastering XMI 2002, S. 330f
[13] Vgl. dazu im Folgenden Object Management Group, MDA Guide 2014

- Platform Specific Model (PSM): Dieses Modell kann aus dem PIM mithilfe einer Transformations-Spezifikation für die entsprechende Plattform generiert werden

Die OMG gibt mit dem MDA-Ansatz die Standards vor, die Entwickler und Hersteller von MDA-Werkzeugen für die modellgetriebene Softwareentwicklung verwenden sollten. Konkrete Implementierungen findet man eher selten, häufig sind aber Teile des Ansatzes umgesetzt. So werden in der Praxis auch meist weniger Modelle und weniger Transformationen benutzt und es wird z. B. der Quellcode direkt aus dem PIM heraus generiert oder es wird nur ein PSM erstellt.

Entscheidend für den Erfolg des MDA-Ansatzes in der Praxis ist auch die Qualität der Generatoren. Für einen sinnvollen Einsatz der Generatoren und zur Durchsetzung des Aufwandes, der mit der Umstellung auf den Ansatz verbunden ist, müssen daher u. a. die folgenden Voraussetzungen erfüllt sein: Generatoren sollten möglichst viele Artefakte automatisch aus dem Modell heraus generieren können. Dabei sollten so viele Informationen wie möglich aus dem Modell berücksichtigt werden. Und die Portierung auf andere Plattformen sollte durch den Austausch von Komponenten des Generators oder durch den Austausch des Generators selbst leicht erreicht werden können.

2.1.3 Transformatoren und Generatoren

Wie bereits in Abbildung 1 zu sehen ist, können viele Artefakte aus einem Modell heraus generiert werden. In der MDA sind dafür die Modell-Transformationen von großer Bedeutung. „Modelltransformationen sind berechenbare Abbildungen eines Quellmodells in ein Zielmodell.“[14]

Zum einen gibt es die Modell-zu-Modell-Transformationen (M2M oder MMT). Hierbei wird ein Modell durch Informationen angereichert oder erweitert. Die Informationen können dabei ebenfalls aus einem oder mehreren Modellen stammen. Eine solche M2M-Transformation ist beispielsweise der Übergang vom PIM zum PSM.

Zum anderen gibt es Modell-zu-Text-Transformationen (M2T). Hierbei ist das Ergebnis ein Text der aus dem Modell generiert wird, z. B. eine Dokumentation. Die

[14] Flores Beltran, et al., Modellgetriebene Softwareentwicklung 2007, S. 70

Generierung des Quellcodes wird in der Praxis ebenfalls eher zu den M2T-Transformationen gezählt, auch wenn der Quellcode als ein Modell interpretiert werden kann.[15]

Für eine automatisierte Transformation werden in beiden Fällen neben dem formalisierten Modell auch formalisierte Transformationsregeln benötigt, z. B. das von der OMG definierte Konzept der Query View Transformation (QVT). Die Transformation, insbesondere M2T, kann allerdings ebenso mithilfe einer Template Engine erfolgen. Dabei werden Vorlagen oder Schablonen, die entsprechende Befehle der Template Engine enthalten, durch diese mit den Modellinformationen belegt. Auf die genaue Funktionsweise der QVT oder der Template Engines soll an dieser Stelle aber nicht näher eingegangen werden.

2.1.4 Vor- und Nachteile durch ein Modell als Ausgangspunkt

Zusammenfassend seien hier noch mal die Vor- und Nachteile der modellgetriebenen Softwareentwicklung aufgezählt, wie sie z. T. bereits zuvor genannt wurden:

Auf der Seite der Vorteile steht allem voran die gesteigerte Produktivität, durch die automatisierte Generierung mehrerer Artefakte aus einem Modell. Einmal erstellte Generatoren können zudem beliebig oft in anderen Projekten wiederverwendet werden. Hat man sich also einmal auf MDSD spezialisiert, kann man so Ressourcen sparen. Die Artefakte haben zudem eine hohe Qualität, da einmal korrekt erstellte Generatoren immer die gleichen Ergebnisse liefern und die Generatoren meist über längere Zeit erprobt sind. Die Ergebnisse sind also verlässlich, reproduzierbar und darüber hinaus validierbar. Durch den Austausch oder der Anpassung von Generatoren bzw. der Transformationen kann man zudem leicht auf andere Technologien oder Plattformen wechseln sowie mehrere Plattformen gleichzeitig bedienen.

Weiterhin sind Modelle auch ohne MDSD ein nützliches Werkzeug, insbesondere in der Entwurfsphase der Softwareentwicklung. Sie ermöglichen ein einheitliches Verständnis vom System und verschaffen einen guten Überblick für alle beteiligten Stakeholder. Da Modelle also sowieso angefertigt werden sollten, kann man sie auch weitergehend nutzen. Durch MDSD werden die Modelle für die Generierung

[15] Vgl. Flores Beltran, et al., Modellgetriebene Softwareentwicklung 2007, S. 70

zwingend aktuell gehalten, was sonst häufig vernachlässigt wird, nachdem die Entwurfsphase abgeschlossen wurde.

Auf der Seite der Nachteile sind zuerst die hohen Ressourcenkosten beim Umstieg oder bei der Einführung der MDSD in bestehende Strukturen zu nennen. Die Einführung erfordert eine Schulung und Einweisung der Mitarbeiter. Darüber hinaus fallen ggf. Lizenzkosten für die verwendeten Technologien und Frameworks an. Die Portierung eines bestehenden Systems auf den modellgetriebenen Ansatz kann zeitintensiv sein und währenddessen erhält das System keine neuen Funktionen. Aus wirtschaftlicher Sicht lohnt sich ein Umstieg nur selten. Idealerweise bietet es sich daher an, bei neuen Projekten von Anfang an auf MDSD zu setzen.

Nutzt man Frameworks oder Generatoren von Drittanbietern ist die Bindung an diese ggf. sehr stark. Von der Qualität dieser hängt daher viel ab. Halten sich z. B. Frameworks nicht an vorgegebene Standards wird ein Austausch schwierig. Das wird insbesondere dann problematisch, wenn die Weiterentwicklung eines Frameworks eingestellt wird. Insgesamt scheint MDSD bzw. MDA im praktischen Einsatz immer noch nicht weit verbreitet. Die OMG gibt zwar Standards vor; diese setzen sich allerdings nur langsam durch.

Abschließend fasst die folgende Abbildung Vor- und Nachteile nochmals kurz zusammen:

Abbildung 2: Vor- und Nachteile des MDSD

2.2 Objektrelationale Abbildung und O/R-Mapper

Die große Herausforderung, egal ob bei code- oder modellgetriebener Softwareentwicklung, ist die persistente Speicherung der Geschäftsdaten. In diesem Kapitel soll ein Überblick über die Schwierigkeiten und über die möglichen Lösungsstrategien gegeben werden.

2.2.1 Das Problem: Objekte können nicht in relationaler Datenbank gespeichert werden

Das Problem bei der Speicherung von Objekten in einer relationalen Datenbank ist die Diskrepanz zwischen der Objektorientierung einerseits und der zeilen- und spaltenbasierten Speicherung in den Tabellen einer relationalen Datenbank andererseits. Diese Diskrepanz ist auch unter dem Namen „Impedance Mismatch“ bekannt.

Bei der Objektorientierung kapseln Klassen zusammengehörige Daten und ihr Verhalten (Eigenschaften und Methoden). Objekte sind konkrete Ausprägungen einer Klasse und haben einen eindeutigen Zustand, der durch die Daten definiert ist. Klassen können in zweierlei Hinsicht in einer Beziehung zueinanderstehen:

- Hierarchisch in Form von Vererbung (Spezialisierung/Generalisierung): eine Klasse kann die Datenstruktur und das Verhalten von einer anderen Klasse erben
- Assoziationen: geben an, wie die konkreten Objekte einer Klasse miteinander eine Beziehung eingehen können

In einer relationalen Datenbank werden Daten in Tabellen abgelegt, welche aus Spalten und Zeilen bestehen. Die Spalten definieren dabei die Datenstruktur und die Zeilen repräsentieren die Datensätze – auch Tupel genannt. Die Tabelle wird dann mathematisch als Relation angesehen, also als Teilmenge eines kartesischen Produktes der Wertebereiche (Domänen), die die jeweiligen Attribute annehmen können.[16]

Eindeutig identifiziert werden die Datensätze über den Primärschlüssel, bestehend aus einer oder mehreren Spalten. Der Primärschlüssel muss dabei eindeutig sein. „Eine Relation hat keine doppelten Tupel, d. h. Zeilen mit exakt den gleichen Werten werden

[16] Vgl. Faeskorn-Woyke, et al., Datenbanksysteme 2007, S. 126ff

unterdrückt."[17] Eine Objektidentität wie in der Objektorientierung gibt es in relationalen Datenbanken aber grundsätzlich nicht.[18] Zwei unterschiedliche Objekte können exakt gleiche Eigenschaften haben, die Referenzen der Objekte zeigen hingegen auf unterschiedliche Speicherbereiche.

Erst mit der Einführung des SQL-Standards SQL2003 wurde das relationale Modell um objektrelationale Aspekte erweitert und so u. a. eine OID (Objekt ID) zur eindeutigen Identifizierung von Objekten eingeführt.[19] Da allerdings selbst der SQL2-Standard (SQL92) aus dem Jahre 1991/92 von vielen Datenbankherstellern noch nicht vollständig oder einheitlich implementiert ist und auch heute die Umsetzung dieser objektrelationalen Erweiterungen bei namhaften Datenbankanbietern selten zu finden ist[20], wird dies hier nicht weiter betrachtet.

Beziehungen zwischen den Relationen basieren also nicht auf einer Objektidentität, sondern auf der Datenbankidentität, welche durch die Primärschlüssel bestimmt wird. Für Beziehungen werden daher die Primärschlüssel der einen Tabelle als Fremdschlüssel-Spalten zur anderen Tabelle hinzugefügt. Um die Daten dabei konsistent und integer zu halten, gibt es das Konzept der referenziellen Integrität die durch Constraints auf den Fremdschlüsseln umgesetzt wird.[21]

Das objektorientierte Modell und das relationale Modell unterscheiden sich in vielerlei Hinsicht. Das größte Problem ergibt sich sicherlich aus den Vererbungshierarchien, also den Spezialisierungs- bzw. Generalisierungsbeziehungen, für die es im relationalen Modell keine Entsprechung gibt. Auch bei den Beziehungen zwischen Objekten gibt es Unterschiede, da die Objektorientierung eine präzisere Definition von Beziehungen und weitere Beziehungsarten erlaubt. Als drittes Problem sei der Unterschied zwischen Objekt- und Datenbankidentität aufgeführt. Natürlich gibt es noch weitere wesentliche Unterschiede, wie z. B. die Tatsache, dass Objekte ihr Verhalten in Methoden kapseln. So beschäftigen sich viele Diagrammtypen der UML

---

[17] Faeskorn-Woyke, et al., Datenbanksysteme 2007, S. 128
[18] Vgl. Müller und Wehr, Java Persistence API 2 2012, S. 25f sowie Grose, Doney und Brodsky, Mastering XMI 2002, S. 51f
[19] Vgl. Faeskorn-Woyke, et al., Datenbanksysteme 2007, S. 302
[20] Vgl. Faeskorn-Woyke, et al., Datenbanksysteme 2007, S. 197f, 298 sowie Geisler, Datenbanken 2014, S. 207f
[21] Vgl. Faeskorn-Woyke, et al., Datenbanksysteme 2007, S. 153, 223 sowie Geisler, Datenbanken 2014, S. 101f

mit dem Verhalten von Objekten. Das Verhalten ist aber für den Datenaustausch nicht relevant[22] und damit ebenfalls nicht für die persistente Speicherung von Objekten. Hier zählt einzig der Zustand, also die Daten eines Objektes.

Auf andere Datenbanksysteme soll in dieser Arbeit nicht eingegangen werden. Relationale Datenbanksysteme sind in der Praxis immer noch am weitesten verbreitet und haben sich als Standard etabliert. Sie bieten eine konsistente und performante Datenspeicherung strukturierter Daten. Objektdatenbanken bzw. objektorientierte Datenbanken würden die hier beschriebenen Probleme zwar lösen, sie konnten sich allerdings am Markt nicht durchsetzen, u. a. da sie nicht die gewünschte Performance liefern. Stattdessen versuchen nun einige Datenbankhersteller ihre relationalen Datenbanksysteme mit objektrelationalen Erweiterungen zu versehen, welche jedoch auch nur allmählich Verwendung finden.

Die neue Generation von Datenbanksystemen[23], auch NoSQL-Datenbanken genannt, die sich gerade im Bereich der Web 2.0-Anwendungen und Big Data immer mehr durchsetzt, eignet sich besonders gut zum Speichern von unstrukturierten und nicht verknüpften Daten. Sie sind i. d. R. performanter als relationale Datenbanken und lassen sich vor Allem besser skalieren, um auch mit großen Datenmengen, wie sie bei den Web 2.0-Anwendungen aufkommen, umzugehen. Dabei sind sie jedoch häufig schemafrei und haben ein anderes Konsistenzmodell als es von Transaktionen aus dem relationalen Datenbankmodell bekannt ist. Die ACID-Eigenschaften (atomar, konsistent, isoliert und dauerhaft) sind also nicht immer erfüllt. In Web 2.0-Anwendungen ist dies meistens ausreichend, bei kritischen Geschäftsanwendungen hingegen nicht, weshalb sie in dieser Arbeit nicht betrachtet werden.

2.2.2 Die Lösung: objektrelationale Abbildung

Als Lösung für das Problem des Impedance Mismatch gibt es die sogenannte objektrelationale Abbildung (engl. object-relational mapping, ORM). Mithilfe dieser Abbildung lassen sich Objekte persistent in einer relationalen Datenbank speichern. Die Datenstruktur der Objekte und ihrer Beziehungen zueinander werden dabei auf die Spalten und Zeilen einer Tabelle in der Datenbank abgebildet. Objekte, die persistent

[22] Vgl. Grose, Doney und Brodsky, Mastering XMI 2002, S. 41
[23] Vgl. dazu im Folgenden Edlich, et al., NoSQL 2011, S. 2ff

in einer Datenbank gespeichert wurden, können dann mithilfe der Abbildung wieder genauso aus der Datenbank ausgelesen werden.

Es muss allerdings beachtet werden, dass die objektrelationale Abbildung keine vollständige Lösung für das Problem des Impedance Mismatch darstellt. Nach Ireland und Bowers[24] werden damit nur die Symptome bekämpft, die durch die Verbindung von Objektorientierung und Relationen entstehen, um eine akzeptable Lösung zu erhalten. Sie betrachten es als komplexes Problem, bei dem mehrere Problemfelder sich gegenseitig beeinflussen.[25] Dies ist vergleichbar mit dem magischen Viereck der wirtschaftspolitischen Ziele; auch diese können – durch die gegenseitige Beeinflussung – nicht gleichzeitig erreicht werden. In dieser Arbeit wird es aber im Wesentlich nur um zwei dieser Problemfelder gehen – der Struktur und der Identität – für welche es akzeptable Lösungen gibt.

An dieser Stelle sei außerdem auf Ted Newards oft zitierte These „Object/Relational Mapping is the Vietnam of Computer Science“[26] verwiesen. Neward vergleicht ORM mit dem Vietnam Krieg, da auch die Arbeit mit ORM gut anfängt und schnell erste Erfolge bringt, sie dann aber zunehmend schwieriger wird und sich immer mehr Probleme auftun, bis man schließlich den Moment verpasst, an dem man besser aufhören sollte. Dies entkräftet jedoch z. T. Fowler[27], in dem er schreibt, dass die Benutzung von ORM immer noch besser ist, als ganz auf diese zu verzichten und das fehleranfällige Mapping selbst zu schreiben. Als Alternative bleibt lediglich die Vermeidung des Problems, was allerdings nur in wenigen Anwendungsfällen möglich ist.

Für die oben genannten Schwierigkeiten bei Beziehungen und Vererbungshierarchien sowie bei der Identität gibt es im ORM also akzeptable Lösungen, die im Folgenden kurz aufgezeigt werden sollen.

---

[24] Vgl. Ireland und Bowers, Exposing the Myth: Object-Relational Impedance Mismatch is a Wicked Problem 2015, S. 22
[25] Vgl. Ireland und Bowers, Exposing the Myth: Object-Relational Impedance Mismatch is a Wicked Problem 2015, S. 23
[26] Vgl. Neward, The Vietnam of Computer Science 2006
[27] Vgl. Fowler, OrmHate 2012

2.2.2.1 Mapping von Vererbungshierarchien

Nach Rehn[28] und Holder et al.[29] gibt es für das Mapping von Vererbungshierarchien drei mögliche Vorgehensweisen:

- Eine Tabelle pro Klasse: Jeder Klasse in der Vererbungshierarchie wird eine Tabelle zugeordnet, welche dann mit Hilfe von Join-Operationen zusammengeführt werden
- Eine Tabelle für jede konkrete Klasse: Felder von abstrakten Klassen werden immer in den Tabellen der konkreten Nachfahren eingeführt, auf Kosten der Normalisierung
- Eine Tabelle pro Vererbungsbaum: Stärkste Denormalisierung durch die Verwendung einer Tabelle für die gesamte Vererbungshierarchie einer Klassenfamilie

Weiter schreiben sie, dass die beste Strategie immer von der jeweiligen Situation abhängt. In der Praxis wird immer eine Kombination dieser Möglichkeiten gewählt, um die bestmöglichste Mischung aus Normalisierung und Performance zu erhalten.

Eine andere Herangehensweise ist die Betrachtung einer einzelnen Klasse statt der gesamten Hierarchie, um so die oben genannten Strategien viel feiner anwenden zu können. Für das sogenannte Table-Mapping gibt es folgende Möglichkeiten:

- Own: Die Klasse erhält eine eigene Tabelle in der Datenbank
- Parent: Die Felder der Klasse werden in der Tabelle des Vorfahren hinzugefügt (bzw. beim ersten Vorfahren, der wieder ein eigenes Table-Mapping hat)
- Child: Die Felder der Klasse werden in jeder Tabelle der Nachfahren immer wieder hinzugefügt (bzw. auch hier in den Nachfahren, die wieder ein eigenes Table-Mapping haben)

2.2.2.2 Mapping von Beziehungen

Zunächst gibt es drei verschiedene Arten von Multiplizitäten, die für das Mapping von Beziehungen betrachtet werden müssen: eins-zu-eins Beziehungen (kurz 1..1), eins-zu-n Beziehungen (1..n) und n-zu-n Beziehungen (n..n). Optionale Beziehung wie 0..1

---

[28] Vgl. dazu im Folgenden Rehn, FoAM 2007

[29] Vgl. dazu im Folgenden Holder, Buchan und MacDonell, Towards a Metrics Suite for Object-Relational Mappings 2008

oder 0..n verhalten sich von der relationalen Abbildung her genauso. Das Gleiche gilt bei konkreten Multiplizitäten wie 2..4, die entweder 0..n oder n..n entsprechen, da das relationale Modell hierfür keine spezielle Lösung anbietet.

1..n-Beziehungen können durch Fremdschlüssel-Spalten gelöst werden; und zwar auf der entgegengesetzten Seite (die zu-n-Seite verweist auf die zu-1-Seite). Bei 1..1-Beziehungen wird ebenfalls eine einfache Fremdschlüsselspalte auf einer der Seiten benötigt. n..n-Beziehungen können nur mithilfe einer weiteren Tabelle (Link- oder Zwischentabelle genannt) aufgelöst werden, die jeweils die Fremdschlüssel für beide Seiten enthält.

2.2.2.3 Objektidentität vs. Datenbankidentität

Die Objektidentität wird über die Position im Speicher bestimmt, während bei der Datenbankidentität der Primärschlüssel eindeutig sein muss. In beiden Fällen ist die Identität – wie der Begriff bereits sagt – eindeutig: Es gibt keine zwei Objekte an der gleichen Speicherposition und ebenso gibt es keine zwei Tabellenzeilen, die in den Primärschlüsselspalten exakt gleiche Werte besitzen. Die Definition des Primärschlüssels ist beim Mapping also von entscheidender Bedeutung. Die einfachste Lösung ist die Verwendung eines künstlichen Schlüssels (engl. surrogate key) als Primärschlüssel. In diesem Schlüssel wird entweder eine fortlaufende Nummer oder eine Unique ID gespeichert. Dies hat den Vorteil, dass Datensätze einfach identifiziert und referenziert werden können, was wiederum die Übersichtlichkeit gerade bei Beziehungen steigert.[30] Ein künstlicher Schlüssel erleichtert damit ebenso die objektrelationale Abbildung. Verwendet man GUIDs (Global Unique IDs) kann man die Datensätze sogar über Tabellen und Datenbankgrenzen hinaus identifizieren. Für eine fortlaufende Nummer bieten einige Datenbankhersteller auch spezielle Datentypen oder sog. Sequenzen an. Der zusätzlich benötigte Speicherplatz für die weitere Spalte[31] kann aus heutiger Sicht vernachlässigt werden.

---

[30] Vgl. Geisler, Datenbanken 2014, S. 147 sowie Faeskorn-Woyke, et al., Datenbanksysteme 2007, S. 87

[31] Vgl. Geisler, Datenbanken 2014, S. 147

2.2.3 Standardisierte Umsetzung durch O/R-Mapper

Da die objektrelationale Abbildung eine aufwendige, komplexe und damit fehleranfällige Implementierung erfordert, haben sich mit der Zeit gewisse O/R-Mapper und ganze Frameworks entwickelt, die diese Aufgaben für den Entwickler übernehmen. Damit hat der Entwickler die Möglichkeit, sich stärker auf die Modellierung und Implementierung der Geschäftslogik zu fokussieren, während die Persistenz von eben diesen Frameworks übernommen wird.

Die Standardisierung der O/R-Mapper ist im Java-Umfeld am weitesten vorangeschritten. Dort spezifiziert die Java Persistence API (JPA) die für die Persistenz benötigten Schnittstellen, welche durch verschiedene JPA-Provider umgesetzt werden. Die JPA definiert u. a. in welcher Form die benötigten Metadaten für das Mapping vorliegen müssen, wie ein Entity Manager den Lebenszyklus der Entitäten regelt oder wie Daten mithilfe der Java Persistence Query Language (JPQL) abgefragt werden können. Die bekanntesten JPA-Provider sind EclipseLink, Hibernate und Apache OpenJPA.[32] O/R-Mapper stehen allerdings auch in vielen anderen Programmiersprachen, mit teils unterschiedlichem Funktionsumfang, zur Verfügung.

Die Kern-Funktion eines O/R-Mapper ist die relationale Abbildung. Darüber hinaus gibt es aber eine Reihe weiterer Funktionen, die ein O/R-Mapper unterstützen kann und auf die im Folgenden kurz eingegangen werden soll. Einen Überblick gewährt vorab Abbildung 3:

[32] Vgl. Müller und Wehr, Java Persistence API 2 2012

Abbildung 3: Wichtige Funktionen eines O/R-Mappers

Unter CRUD versteht man die grundlegenden Operationen bei der Arbeit mit Datenbanksystemen: Create, Read, Update und Delete. Sie bilden das Minimum des Funktionsumfangs eines O/R-Mappers bei der Arbeit mit den Daten und entsprechen den SQL-Funktionen INSERT, SELECT, UPDATE und DELETE.

Fast ebenso wichtig sind Abfragen (engl. Queries) auf den Daten, wie man sie bei relationalen Datenbanken in Form der SQL SELECT-Anweisung kennt. Abfragen sind vereinfacht ausgedrückt beliebig komplexe Ausdrücke, um Daten zu erhalten oder Tatbestände zu prüfen. Die erfragten Daten können dabei zusätzlich vorgefiltert und sortiert werden. Die Abfragen werden häufig durch eine eigene Abfragesprache realisiert, die der SQL der relationalen Datenbanken ähnlich ist, allerdings auf den Objekten statt auf Relationen basiert.

Der Punkt Navigation bezieht sich auf Assoziationen. Hier ist es wichtig, ob der O/R-Mapper die Objekte beim Zugriff auf eine Assoziation automatisch lädt, oder ob dies vom Entwickler manuell ausgeführt werden muss. Zudem kann sowohl bei Assoziationen wie auch bei Attributen unterschieden werden, ob diese sofort mit den

Daten des Objektes geladen werden sollen (sog. Eager Loading) oder erst bei Bedarf, also beim Zugriff auf diese (sog. Lazy Loading).

Transaktionen sorgen dafür, dass sich die Daten immer in einem konsistenten Zustand befinden, indem die Änderungen innerhalb einer Transaktion entweder zusammen oder überhaupt nicht durchgeführt werden. Sie stimmen damit mit den SQL-Transaktionen überein und besitzen ebenso die ACID-Eigenschaften.[33] Es gibt einfache Transaktionen, geschachtelte Transaktionen (Transaktion innerhalb einer Transaktion) und parallele Transaktionen (mehrere Transaktionen nebeneinander).

Um Daten nur einmal aus der Datenbank zu laden, kann ein Zwischenspeicher (engl. Cache) verwendet werden, welcher die geladenen Daten im Speicher vorhält. Es sind ebenso zwischengespeicherte Abfragen möglich, damit die dahinterstehende SQL-Abfrage nicht immer neu gebildet werden muss.

Viele O/R-Mapper unterstützen auch den gleichzeitigen Zugriff auf die Daten von mehreren Benutzern unter Wahrung der Konsistenz. Hierbei spielen zudem die Locking-/Concurrency-Mechanismen zum Sperren von Daten bzw. zusammenhängenden Daten eine wichtige Rolle. Eine weitere Stufe darüber liegt die Unterstützung für mehrere Mandanten, sog. Multitenancy-Systeme. Dies ist insbesondere bei Software as a Service (SaaS) wichtig und sorgt dafür, dass mehrere Kunden mit dem gleichen System arbeiten, aber jeweils ein eigener privater Datenbestand zugrunde liegt.

Schließlich gibt es noch die Möglichkeit Objekte zu versionieren, also alte Historienstände der Datensätze ebenfalls in der Datenbank vorzuhalten. Dazu wird z. B. bei einer Änderung nicht der Datensatz in der Datenbank aktualisiert, sondern ein neuer Datensatz eingefügt. Zusätzlich wird dann eine weitere Spalte benötigt, in der eine Versionsnummer oder ein Zeitstempel gespeichert wird.

[33] Vgl. Faeskorn-Woyke, et al., Datenbanksysteme 2007, S. 443f

2.3 Modellgetriebene O/R-Mapper

Modellgetriebene O/R-Mapper sind spezielle O/R-Mapper, die im Sinne der MDA ein Modell als Ausgangspunkt haben.[34] In diesem Kapitel sollen die Unterschiede zu herkömmlichen O/R-Mappern aufgezeigt werden.

2.3.1 Verbindung von modellgetriebener Softwareentwicklung und O/R-Mappern

Ein O/R-Mapper im engeren Sinne behandelt lediglich das Mapping selbst, also die objektrelationale Abbildung. Ein O/R-Mapper im weiteren Sinne umfasst häufig auch weitere Funktionen, die mit dem Mapping in Berührung kommen, dabei aber über das reine ORM hinausgehen. In diesem Fall kann man dann von einem O/R-Mapping-Framework sprechen. Für diese Arbeit sind zwei Kategorien von O/R-Mappern i. w. S. zu unterscheiden: code- und modellgetriebene O/R-Mapper.[35]

Codegetriebene O/R-Mapper sind weit verbreitet und zu ihnen zählen u. a. die bekannten O/R-Mapper EclipseLink oder Hibernate für Java, NHibernate für das .Net-Umfeld und Doctrine für PHP. Sie folgen i. d. R. dem „Code First“-Ansatz, bei dem zuerst der Quellcode geschrieben wird – je nach Anwendungsfall mit oder ohne Logik und häufig mit einfachen Objekten.[36] Oft bieten diese O/R-Mapper auch die Möglichkeit, die Klassenstruktur durch die Analyse eines bestehenden Datenbankschemas zu erzeugen („Database First“-Ansatz). Dadurch sind sie insbesondere bei der Anpassung bzw. Weiterentwicklung von bestehenden Legacy Systemen nützlich. Dieser Quellcode wird dann mithilfe von Metadaten (z. B. Annotationen oder XML-Dateien) an den O/R-Mapper gebunden. Logik und Persistenz können so sauber voneinander getrennt und die Klassen bzw. die Geschäftslogik können ohne den O/R-Mapper zusätzlich in anderen Szenarien genutzt werden. Dies ist z. B. für Unittests praktisch.

Für modellgetriebene O/R-Mapper bildet ein (UML-)Modell den Kern des Anwendungssystems und den Ausgangspunkt der Entwicklung, die nach dem MDA-Ansatz stets vom Modell über den Quellcode zur Persistenz führt. Dieser Ansatz bietet

---

[34] Vgl. dazu und im Folgenden Herrmann, Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper 2015, S. 15f

[35] Vgl. dazu im Folgenden Herrmann, Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper 2015, S. 13f

[36] In Java auch bekannt als POJO (Plain Old Java Object).

– wie für die MDA üblich – einen hohen Automatisierungsgrad und eine daraus folgende Produktivitätssteigerung.[37] Neben den bereits genannten Artefakten wie dem Quellcode können überdies die für das ORM benötigten Metadaten sowie das Datenbankschema direkt aus dem Modell heraus generiert werden. Auf die Verwendung von normalen Objekten muss dabei jedoch bei vielen O/R-Mappern dieser Art verzichtet werden. Eine Abwandlung des modellgetriebenen Ansatzes sieht sogar vor, ganz auf die Generierung von Programmcode zu verzichten und das Modell direkt zur Laufzeit auszuführen[38], was insbesondere beim Thema Prototyping und Tests hilfreich ist. Diese Ausführung von Modellen wird auch Executable UML oder kurz xUML genannt. Modellgetriebene O/R-Mapper bieten sich insbesondere bei Neuentwicklungen an, bei denen das zugrundeliegende Modell stetig wächst. Die Weiterentwicklung von Legacy Systemen ist hier aufwendig und automatisierte Umwandlungen eines bestehenden Datenbankschemas in ein Modell sind eher selten zu finden. Beispiele für modellgetriebene O/R-Mapper sind Athena, GeneSEZ und Texo (Java) sowie ECO und nHydrate (C#/.Net).

2.3.2 Besonderheiten im Detail

Durch den modellgetriebenen Ansatz bieten entsprechende O/R-Mapper häufig einen Funktionsumfang, der deutlich über den normaler O/R-Mapper i. e. S. hinausgeht. Sie sind daher eher als Frameworks zu betrachten, denn als einzelne Komponente. Im Folgenden wird daher auch von modellgetriebenen O/R-Mapping-Frameworks oder kurz O/R-Framework gesprochen.

Die modellgetriebenen O/R-Frameworks beherrschen i. d. R. auch alle Disziplinen des O/R-Mappings wie sie in Kapitel 2.2.3 beschrieben wurden. Die Metadaten und Mapping-Einstellungen werden hingegen bereits im Modell beschrieben und zwar auf eine standardisierte Weise. Je nach Umsetzung kann dann der O/R-Mapper direkt auf diese Metadaten zugreifen oder – wenn er abgekoppelt als eigenständige Komponente arbeitet – können die benötigten Mapping-Informationen aus dem Modell heraus generiert werden. Letzteres erfolgt dann entweder mit dem Quellcode zusammen in Form von Annotationen oder in einer gesonderten Datei. Das Modell kann ebenso für

[37] Vgl. nHydrate, nHydrate 2010
[38] Vgl. Mellor und Balcer, Executable UML: a foundation for model-driven architecture 2002, S. 5ff, Liddle, Model-Driven Software Development 2010, S. 18 sowie Microsoft Corporation, Design Time Code Generation and Runtime Model-Driven Generation 2010

eine Entscheidung über die verwendete Mapping-Strategie der Vererbung (s. Kapitel 2.2.2.1) herangezogen werden. Die wahrscheinlich beste Strategie kann sogar anhand von diverser Metriken, die aus dem Modell gebildet werden, berechnet werden, wie das Konzept von Holder et al.[39] zeigt.

Existiert die Datenbank bzw. das Datenbankschema noch nicht, können O/R-Mapper dieses Schema in Form von DDL-Befehlen (Data Definition Language) generieren. Die modellgetriebenen Pendants können das ebenfalls. Darüber hinaus ist aber mithilfe des Modells eine Aktualisierung des Datenbankschemas wesentlich einfacher möglich. Insbesondere wenn das Modell einer Versionskontrolle unterliegt, lassen sich die benötigten Änderungen am Datenbankschema leicht durch einen Vergleich der Modellversionen berechnen. Der Datenverlust oder der manuelle Anpassungsaufwand der DDL-Skripte ist dadurch minimal. Diese Technik wird in dieser Arbeit auch als „Database Evolution“ bezeichnet.

Durch das Modell als Basis lassen sich auch weit mehr als leere Entitäten generieren. Ein Modell bietet grundsätzlich die Möglichkeit neben der Struktur auch das Verhalten zu bestimmen. Dazu werden häufig Zustands- oder Aktivitätsdiagramme verwendet, die sich z. T. sogar in Quellcode übersetzen lassen. Da gerade der Zustand eines Objektes eng mit den Daten verbunden ist, macht es daher durchaus Sinn, wenn ein modellgetriebener O/R-Mapper auch um den Zustand der Objekte weiß.

Darüber hinaus sind weitere Artefakte wie z. B. Dokumentationen aus dem Modell heraus generierbar. Diese haben allerdings nicht mehr viel mit ORM zu tun, weshalb an diese Stelle nicht weiter darauf eingegangen wird.

Abschließend kann ein (Meta-)Modell auch validiert bzw. auf Fehler überprüft werden. Die Validierung kann sich dabei ebenso auf die O/R-Mapping-Einstellungen erstrecken. Es dürfen z. B. keine zwei Klassen oder Attribute auf die gleiche Tabelle bzw. Spalte abgebildet werden. Ferner dürfen keine von der Datenbank reservierten Bezeichner verwendet werden.

[39] Vgl. Holder, Buchan und MacDonell, Towards a Metrics Suite for Object-Relational Mappings 2008

2.3.3 Vor- und Nachteile gegenüber herkömmlichen O/R-Mappern

Bei den Vorteilen steht die bereits beschriebene Produktivitätssteigerung durch die Automatisierung im Vordergrund. Die Mapping-Einstellungen, die bei codegetriebenen O/R-Mappern von Hand erstellt werden müssen (z. B. mit Annotationen oder XML-Dateien), können nun aus dem Modell heraus generiert werden. Ebenso lässt sich die Tabellenstruktur der Datenbank generieren, wobei diese Fähigkeit ebenso den codegetriebenen O/R-Mappern zur Verfügung steht. Darüber hinaus gelten natürlich die gleichen Vorteile, die bereits in Kapitel 2.1.4 allgemein für den MDSD-/MDA-Ansatz beschrieben wurden. So gilt auch hier, dass immer qualitativ gleichbleibende und verifizierbare Ergebnisse erzielt werden.

Auf der Seite der Nachteile ist u. a. die geringere Flexibilität zu nennen. Nicht jede spezielle Einstellung eines O/R-Mappers lässt sich immer durch ein Modell abbilden. Besonderheiten eines O/R-Mappers oder eine spezielle Funktion können möglicherweise nicht genutzt werden. Oder sie sind nur durch individuelle Implementierungen im Modell zu lösen, wodurch die Persistenz jedoch nicht mehr klar vom Modell bzw. der Geschäftslogik zu trennen ist. Verwendet man beim modellgetriebenen Ansatz also standardisierte Komponenten, können diese nicht immer in vollem Umfang genutzt werden. Wird hingegen ein Framework verwendet, bei dem alle Funktionen integriert sind, ist man an den Funktionsumfang des Frameworks gebunden. Dieser liegt aber möglichweise hinter dem Umfang einzelner Komponenten, die sich auf eine Sache spezialisiert haben. Weiterhin gibt es teilweise auch Schwierigkeiten, wenn es darum geht eine Anwendung zu modularisieren.[40]

[40] Vgl. Herrmann, Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper 2015, S. 14

3 Konzeption eines Vergleichs ausgewählter O/R-Mapper

In diesem Kapitel geht es um die Konzeption eines Vergleichs von modellgetriebenen O/R-Mapping-Frameworks, welcher dann im nächsten Kapitel durchgeführt wird. Zum einen sollen Ziele und Herangehensweise des Vergleichs beschrieben werden (Kapitel 3.1) und zum anderen sollen die zu vergleichen O/R-Mapper ausgewählt (Kapitel 3.2) und die Vergleichskriterien bestimmt werden (Kapitel 3.3).

3.1 Ziele und Herangehensweise des Vergleichs

Für einen sinnvollen Vergleich ist es wichtig, zu Beginn die Ziele des Vergleiches zu definieren. So können der Ansatz und die Herangehensweise an den Vergleich bestmöglich gewählt werden.

Im Fokus des Vergleiches sollen modellgetriebene O/R-Mapper bzw. O/R-Frameworks stehen, da auf diesen das Hauptaugenmerk dieser Arbeit liegt. Damit ist gleichzeitig die Auswahl der O/R-Mapper auf eine kleine Anzahl beschränkt und passt in den Umfang dieser Arbeit.

Das Hauptziel soll also die Herausarbeitung der Gemeinsamkeiten und Unterschiede verschiedener modellgetriebener O/R-Mapping-Frameworks sein. Neben den Unterschieden im Funktionsumfang ist es interessant, die vom jeweiligen O/R-Framework verwendeten Konzepte auf Gemeinsamkeiten zu untersuchen. Möglicherweise haben sich hier Standards durchgesetzt, die bei mehreren Frameworks wiederzufinden sind.

Für den Vergleich im nächsten Kapitel wurde deshalb eine alternierende Gliederung gewählt. Auf diese Weise wird bereits durch die Gliederung der Blick auf die einzelnen Vergleichskriterien gelenkt und damit eine genaue Analyse bezogen auf jeweils ein Kriterium erreicht. Bei einer Blockgliederung bestünde hingegen die Gefahr, lediglich die verschiedenen O/R-Frameworks zu beschreiben.[41]

Darüber hinaus soll auch eine Wertung der verschiedenen Frameworks erfolgen, um gegebenenfalls Empfehlungen für oder gegen ein gewisses Framework aussprechen zu können. Diese Bewertung sollte dabei abhängig vom Einsatzgebiet erfolgen: Z. B. benötigt eine kleine Anwendung, welche nur von einem Benutzer bedient wird, kein

[41] Vgl. Kornmeier, Wissenschaftlich schreiben leicht gemacht 2013, S. 162

O/R-Framework mit dem maximalen Funktionsumfang, sondern eher eine einfache, leichtgewichtige und performante Lösung. Insgesamt soll die Bewertung aber keinen großen Teil des Vergleiches einnehmen, da es im Bereich der O/R-Mapper regelmäßig Änderungen gibt und z. B. neue O/R-Mapper auftauchen. Vielmehr soll der Leser nach dem Studium des Vergleiches in der Lage sein, abhängig vom Funktionsumfang, das für sich geeignete O/R-Framework auszuwählen. Deshalb sollen auch nur einige wenige ausgewählte Frameworks verglichen werden. Der Vergleich bleibt so überschaubar, gewährt allerdings einen Einblick in die Gemeinsamkeiten und Unterschiede. Keinesfalls soll der Vergleich einen Anspruch auf Vollständigkeit erheben.

Für den Vergleich wurde außerdem ein Beispielmodell erstellt, um ein möglichst einheitliches Bild zu erhalten und auf das man sich während des Vergleichs beziehen kann. Es handelt sich um ein einfaches UML-Modell mit einem Klassendiagramm. Die Sinnhaftigkeit stand dabei nicht im Vordergrund, sondern lediglich die Anforderung, möglichst viele unterschiedliche UML-Elemente und Datentypen in einem kompakten Modell zu verwenden. Ebenso erhebt das Beispiel keinen Anspruch auf Korrektheit. Das Modell wurde mit der Software Modelio erstellt. Einen Überblick bietet die folgende Abbildung 4, das vollständige Modell befindet sich im Anhang.

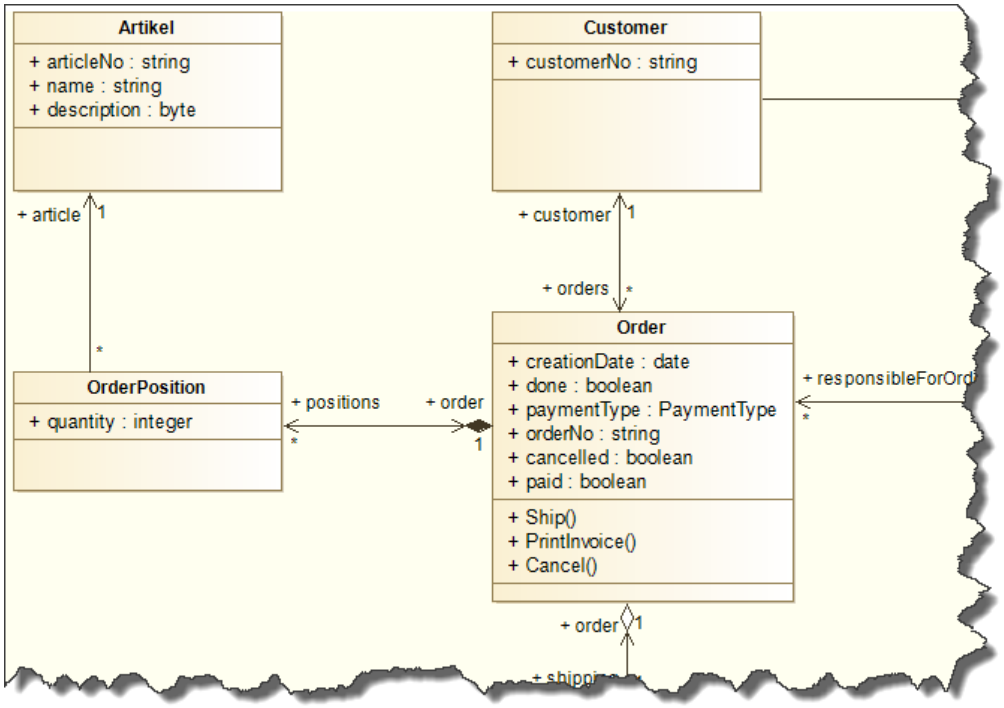


Abbildung 4: Beispielmodell (Ausschnitt)

3.2 Auswahl der zu vergleichenden O/R-Mapper

Der Vergleich sollte sich möglichst über O/R-Frameworks aus verschiedenen Programmiersprachen erstrecken, denn vielleicht beeinflusst die Programmiersprache auch den Funktionsumfang eines O/R-Mappers. Eventuell werden gewisse Funktionalitäten in anderen Programmiersprachen anders gelöst. Außerdem spielt die Art der Programmiersprache eine wichtige Rolle, wenn es um die Performance der Anwendung geht. Ob es sich um einen O/R-Mapper für eine in nativen Maschinencode kompilierende Sprache oder für eine Sprache, die zur Laufzeit von einer virtuellen Maschine interpretiert wird, handelt, dürfte sicherlich Einfluss darauf nehmen. Die Performance spielt bei diesem Vergleich aber nur eine untergeordnete Rolle, da die Vergleichsobjekte sehr unterschiedlich sind und einen Vergleich in dieser Hinsicht schwierig machen. Ein Vergleich der Performance wäre nur möglich, wenn ein Produkt mehrere Programmiersprachen unterstützt.

Ein weiteres Kriterium für die Auswahl ist die Popularität des O/R-Frameworks, z. B. gemessen an der Anzahl von Downloads oder der Erwähnung in der Fachliteratur. Die zu vergleichenden O/R-Mapper sollten idealerweise eine große Verbreitung haben und keine Nischenprodukte sein. Eine hohe Popularität weist meist auf ein stabiles Produkt mit weniger Fehlern hin. In der Regel sind die O/R-Mapper am populärsten, die als Open Source zur Verfügung stehen und somit von der Entwicklergemeinde gepflegt und weiterentwickelt werden können. Open Source Software erleichtert auch den Vergleich selbst. Sie kann einfacher verglichen werden – nicht nur durch die Möglichkeit, sich Implementierungsdetails anzuschauen, sondern auch aus Gründen der Beschaffung – da sie frei verfügbar ist. Allerdings soll gleichermaßen ein Blick auf die Leistung und das Potenzial bekannter kommerzieller Produkte gewährt werden.

Ursprünglich sollten in dieser Arbeit sechs unterschiedliche O/R-Mapping-Frameworks miteinander verglichen werden, die ihren Fokus auf die Programmiersprachen Java, C# und Delphi legen. Java und C# gehören nach Meinung des Autors derzeit zu den beliebtesten und meistgenutzten Programmiersprachen. Das spiegelt sich auch am Arbeitsmarkt wieder: Kenntnisse in einer dieser Sprachen sind fast immer Voraussetzung. Beide Sprachen sind bereits längere Zeit am Markt etabliert und gehören damit nicht zu den kurzweilig aufkommenden Modesprachen. Ebenso wenig Delphi. Delphi entstand in etwa zur gleichen Zeit wie Java und ist heute zwar nicht mehr so stark verbreitet, wird jedoch noch oft für performante

Datenbankanwendungen eingesetzt. Im Gegensatz zu Java und C# wird der Quellcode vom Delphi Compiler direkt in nativen Maschinencode und nicht in einen Bytecode übersetzt, der dann zur Laufzeit von einer virtuellen Maschine weiter übersetzt werden müsste. Für einen ausgewogenen Vergleich waren zwei Frameworks pro Programmiersprache vorgesehen - je ein leichtgewichtiges, schlankes und ein umfangreiches, komplexes Framework.

Dieses Ziel konnte jedoch nicht erreicht werden, da nicht genügend lauffähige und zu dieser Arbeit passende Frameworks gefunden werden konnten.

So sind einige O/R-Mapper zwar modellgetrieben, jedoch basieren sie nicht auf einem UML-Modell und dem MDA-Ansatz, auf den im ersten Teil dieser Arbeit hingearbeitet wurde, sondern auf einem ER-Modell (Entity-Relationship-Model, ERM). Als Beispiele sind hier die O/R-Mapper EntityFramework (für das .Net Framework, von Microsoft), nHydrate (Open Source Aufsatz für das Entity Framework[42]) und EntityDAC (kommerzielle O/R-Lösung für Delphi[43]) zu nennen.

Wieder andere entsprechen zwar den Voraussetzungen, werden allerdings mittlerweile nicht mehr unterstützt oder sind teilweise nicht mehr lauffähig. Hier sind als Beispiele die Generator-Frameworks AndroMDA[44] sowie GeneSEZ[45] zu nennen. Beide sind von den Konzepten her gut geeignete und mächtige Frameworks für den MDA-Ansatz, kommen aber leider ohne aktuell funktionsfähige Generatoren für das O/R-Mapping für diesen Vergleich nicht in Frage. Ebenso vielversprechend ist das Athena Framework mit eigenem O/R-Mapper[46], welches auch nicht mehr vollständig funktionsfähig ist.

Im Ergebnis fiel die Auswahl für den Vergleich in dieser Arbeit daher auf nur noch drei Frameworks - je eines für die o. g. Programmiersprachen Java, C# und Delphi - deren Auswahl im Folgenden mit einer kurzen Vorstellung begründet werden soll.

---

[42] Vgl. nHydrate, GitHub - nHydrate 2016
[43] Vgl. Devart, ORM for Delphi with LINQ Support 2016
[44] S. AndroMDA.org, AndroMDA Model Driven Architecture Framework – AndroMDA - Homepage 2014
[45] S. Generative Software Engineering Zwickau, GeneSEZ: Welcome to GeneSEZ o. J.
[46] S. AthenaSource, Athena Framework for Java Overview - Athena Framework o. J.

3.2.1 Texo & EclipseLink

Texo gehört zu den EMFT-Produkten der Eclipse Foundation. Das EMF (Eclipse Modeling Framework) ist die standardisierte Lösung zur Modellierung in Eclipse. Es handelt sich dabei im Kern um ein eigenes Metamodell, welches z. B. UML-Modelle nutzen kann, um aus diesen Java-Quellcode zu generieren.[47] Es kann dabei direkt in die Eclipse-IDE eingebunden werden.

Texo[48] nutzt das EMF, um daraus nicht nur Java-Quellcode, sondern auch das ORM/JPA-Mapping beispielsweise für den O/R-Mapper EclipseLink zu generieren. Ebenso stellt es Modellinformationen zur Laufzeit zur Verfügung. Texo ist der Nachfolger des nicht mehr weiterentwickelten Teneo-Frameworks und hat sich im Gegensatz zu diesem auf Webdienste spezialisiert. Simple Webdienste, z. B. für Abfragen auf einer Modellinstanz, können so fast vollständig automatisiert generiert werden.

3.2.2 MDriven

Das MDriven Framework (ehem. ECO) ist die Rundum-MDA-Lösung der Firma CapableObjects.[49] Mit dem eigenem O/R-Mapper lassen sich verschiedene Datenbanksysteme nutzen und mit dem integrierten Modell-Designer lassen sich auch die Benutzeroberflächen für unterschiedliche Systeme im Modell konfigurieren.

Als Zielsystem wird allerdings nur das .Net-Framework aus dem Hause Microsoft unterstützt. Zudem ist es der Nachfolger des nachfolgend vorgestellten Bold for Delphi.

3.2.3 Bold for Delphi

Bold for Delphi[50] ist das erste kommerziell erfolgreiche MDA-Framework für die Programmiersprache Delphi und auch nahezu das einzige. Es wird mittlerweile nicht mehr offiziell weiterentwickelt, befindet sich aber selbst heute noch im Einsatz. Der Vorgänger des ebenfalls hier vorgestellten MDriven/ECO bietet ebenfalls eine flexible Auswahl des Datenbanksystems, allerdings keine modellierte Benutzeroberfläche.

---

[47] Vgl. Eclipse Foundation, Eclipse Modeling Project o. J.
[48] Vgl. dazu im Folgenden Eclipse Foundation, Texo - Eclipsepedia 2015
[49] Vgl. dazu im Folgenden CapableObjects AB, MDriven Framework – Model driven framework formerly known as ECO | CapableObjects o. J.
[50] S. BoldSoft MDE AB, Bold for Delphi 2002

Die letzte offizielle Version von Bold for Delphi stammt aus dem Jahre 2006 und der Support wurde eingestellt. Dies und die Tatsache, dass Bold for Delphi nicht Open Source ist, sprichen daher eigentlich gegen die zuvor genannten Kriterien für die Auswahl der O/R-Frameworks. Die Erfahrungen des Autors bieten hier aber einen tieferen Einblick und zudem lassen sich dank des direkten Vergleiches mit dem Nachfolger auch Fortschritte im Bereich der O/R-Mapper und beim MDA-Ansatz allgemein sowie Unterschiede zwischen den beiden Programmiersprachen Delphi und C# feststellen.

3.3 Auswahl der Vergleichskriterien

Um zunächst einen Überblick über alle O/R-Mapping-Frameworks zu erhalten, sollte der Vergleich mit einem Überblick über die allgemeinen Merkmale der Frameworks beginnen. Dazu gehört Allem voran die Programmiersprache. Daneben ist es wichtig zu wissen, welche Ziele das Framework verfolgt, um so die möglichen Einsatzgebiete abzustecken und um das Framework besser einordnen zu können. Neben der Prioritätensetzung wie z. B. Performance oder Speicherbedarf ist hierbei der Funktionsumfang entscheidend. Zuletzt soll bei den allgemeinen Merkmalen auch auf die Lizensierung und die Kosten eingegangen werden.

Das nächste Vergleichskriterium widmet sich detaillierter dem Funktionsumfang. Bei jedem O/R-Mapping-Framework sollen die wichtigsten Funktionen aufgezeigt werden, mit dem o. g. Hauptziel, Gemeinsamkeiten und Unterschiede sowie Alleinstellungsmerkmale der verschiedenen Frameworks zu ermitteln.

Bei den folgenden Vergleichskriterien rückt der Fokus dieser Arbeit mehr in den Vordergrund. Es wird der Art und Weise, wie der MDA-Ansatz im jeweiligen O/R-Mapping-Framework umgesetzt wird, besondere Aufmerksamkeit beigemessen. Dabei spielt die Nutzung von Standards ganz allgemein eine wichtige Rolle, weshalb dies auch ein eigenes Vergleichskriterium sein wird. Besonders Augenmerk soll auf die Konformität zur UML gelegt werden. Positiv wäre hier die Nutzung von UML mit wenigen individuellen Erweiterungen, sodass das Framework ggfs. einfach ausgetauscht werden kann, während das UML-Modell als Kern des Systems erhalten bleibt.

Als letzter Punkt beim Vergleich sollen die O/R-Mapper im Hinblick auf die Modularisierbarkeit untersucht werden. Dies ist gerade bei umfangreichen und komplexen Anwendungen wichtig, mit modellgetriebenen O/R-Mapper aber nicht leicht umsetzbar.[51]

[51] Vgl. Herrmann, Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper 2015, S. 14

4 Vergleich ausgewählter modellgetriebener O/R-Mapping-Frameworks

Dieses Kapitel umfasst den eigentlichen Vergleich der ausgewählten O/R-Mapping-Frameworks. Die Gliederung erfolgt alternierend nach den Vergleichskriterien. Die Vergleichsobjekte sind jeweils ohne eine Wertung alphabetisch sortiert. Eine Bewertung der Vergleichsergebnisse folgt dann in Kapitel 5.

4.1 Allgemeine Merkmale

Der Vergleich beginnt mit den allgemeinen Merkmalen der jeweiligen O/R-Frameworks. Damit soll eine erste grobe Einordnung der Frameworks ermöglicht werden.

Zu den allgemeinen Merkmalen gehört in diesem Vergleich zunächst die Programmiersprache. Die Programmiersprache gibt bereits einen Einblick in die möglichen Anwendungsszenarien, die üblicherweise mit der jeweiligen Programmiersprache umgesetzt werden. So werden z. B. mit Java bzw. Java EE i. d. R. plattformunabhängige Webdienste entwickelt, während bei Delphi die native Programmierung für Windows im Vordergrund steht, häufig noch mit einer Client-Server-Architektur. Darüber hinaus haben auch die Programmiersprachen selbst einen unterschiedlichen Funktionsumfang, welcher gegebenenfalls Einschränkungen für den O/R-Mapper mit sich bringen könnte. Falls ein Framework mehrere Sprachen unterstützt, sollte dies hier ebenfalls hervorgehoben werden. Die nachfolgenden Untersuchungen werden dann allerdings zur Vereinfachung nur auf eine Programmiersprache bezogen.

Ein weiteres zu vergleichendes Merkmal ist eine Übersicht über den Produktumfang und die Komplexität, die damit ggf. einhergeht. Hierbei soll es nicht um den Funktionsumfang im Detail gehen (siehe dazu das nächste Kapitel), sondern lediglich um eine Einschätzung. Ein leichtgewichtiges Framework bietet meist nur die am häufigsten benötigten Funktionen und erlaubt damit selbst Einsteigern schnelle Erfolge. Darüber hinaus kann weniger Ballast von nicht benötigten Funktionen auch eine bessere Performance bedeuten. Ein umfangreiches Framework hingegen bietet den größtmöglichen Funktionsumfang, jedoch zu Lasten der steigenden Komplexität. Dafür bekommt man aber i. d. R. eine bessere Flexibilität und eine hohe Anpassungsfähigkeit. Diese Frameworks lassen sich meist für jedes Szenario nutzen.

Aus dieser Einschätzung aufbauend, soll der Vergleich dann mögliche Einsatzgebiete und Anwendungsszenarien aufzeigen. Dies können z. B. große mehrschichtige Enterpriseanwendungen mit freier Skalierbarkeit oder einfache lokale Anwendungen für nur einen Benutzer sein. Damit lassen sich dann Aussagen zur Zielgruppe des O/R-Frameworks treffen.

Außerdem ist hier wichtig, wie einfach oder kompliziert sich die Installation und Einrichtung gestaltet. Um Anfängern den Einstieg zu erleichtern, sollte dies zum einen gut dokumentiert sein und zum anderen möglichst einfach und schnell ablaufen. Danach ist die Qualität von Hilfen und Anleitungen sowie Beispielprojekten entscheidend, um tiefer in die Materie einsteigen zu können. Auch eine vollständige Dokumentation des gesamten Funktionsumfanges wäre hier positiv. Weiterhin wäre eine – wenn auch eher subjektive – Einschätzung zur Anzahl von weiteren Ressourcen aus der Entwicklergemeinde hilfreich. Damit zeigt sich, ob man als Anwender leicht Hilfe erhalten kann, da viele andere das Framework ebenso nutzen.

Als letzten Punkt sind aus wirtschaftlicher Sicht die Lizensierung und die damit verbundenen Kosten zu vergleichen. Hier stehen sich frei verfügbare Open Source-Lösungen und kostenpflichtige Closed Source-Lösungen gegenüber. Bei den kostenpflichtigen Angeboten muss zudem unterschieden werden, ob das Lizenzmodell eine einmalige oder regelmäßige fortlaufende Zahlung für einen Softwaredienst (SaaS) vorsieht.

Tabelle 1 fasst die Ergebnisse, die im Folgenden im Detail beschrieben werden, für einen kurzen Überblick zusammen:

Tabelle 1: Überblick über die allgemeinen Merkmale der verglichenen O/R-Frameworks

Framework	*Sprache*	*Umfang/ Komplexität*	*Einsatzgebiete/ Zielgruppen*	*Installation & Einrichtung*	*Hilfe, Anleitungen, Dokumentation*	*Lizensierung & Kosten*
Bold	Delphi	Mittel bis Groß	Überwiegend native Client-Server-Anwendungen mit Fat Clients	Einfach und schnell mittels Installer; < 45 Minuten bis zum Beispielprojekt	Fast vollständige Dokumentation aller Komponenten, einige Tutorials, im Internet ist nicht mehr viel zu finden	Closed Source, kostenpflichtig (nicht mehr gesondert erhältlich)
MDriven	C#	Groß	Klassische Client-Server-Anwendungen und Webdienste	Einfach und schnell mittels Installer; < 30 Minuten bis zum Beispielprojekt	Fast vollständige Dokumentation aller Komponenten, viele Beispielprojekte, noch aktive Community	Teilweise Closed Source, kostenpflichtig
Texo	Java	Mittel bis Groß	Webdienste	Aufwendiger, Eclipse-Plug-In, Beispielprojekt über Git, Unterstützung für Maven; 2 Stunden bis zum Beispielprojekt	Quellcode mit Javadoc fast vollständig dokumentiert, Wiki mit Erläuterungen der wichtigsten Konzepte, aktueller Blog	Open Source

4.1.1 Bold for Delphi

Bold for Delphi ist ein MDA-Framework und O/R-Mapper für das Borland Developer Studio, erstmals veröffentlicht von der schwedischen Firma BoldSoft AB im Jahre 1997.[52] Die Bold Architektur sah ebenfalls einen Quellcode-Generator für C++ vor (Bold for C++), welcher sich gegenüber der Delphi-Variante allerdings nicht durchsetzen konnte. Nachdem Bold von Borland aufgekauft wurde, gab es 2005 die letzte offizielle Version von Bold for Delphi für das Borland Developer Studio 2006.[53] Auch für diese Arbeit liegt der Schwerpunkt bei der Betrachtung dieses Frameworks daher auf der Programmiersprache Delphi. Bei Delphi handelt es sich um eine in nativen Maschinencode kompilierende Sprache. Obwohl Delphi mittlerweile für viele Plattformen zur Entwicklung genutzt werden kann (Windows, MacOS, Android) unterstützt Bold for Delphi lediglich die klassische Windows Programmierung auf 32 Bit-Basis.

Zum Zeitpunkt der Veröffentlichung bot das Framework bereits einen großen Funktionsumfang. Es war dabei von Anfang an auf den MDA-Ansatz der OMG ausgerichtet. Der Schwerpunkt lag auf umfangreichen und komplexen Geschäftsmodellen, die in einer n-Schichtenarchitektur umgesetzt werden konnten. Neben der Quellcode-Generierung beinhaltet das Framework einen O/R-Mapper mit Unterstützung für mehrere SQL-Datenbanksysteme sowie Konzepte für die Präsentation der Geschäftsobjekte in einer Benutzeroberfläche unter Nutzung der OCL (Object Constraints Language). Aus dem Kontakt mit heute noch aktiven Nutzern des Frameworks ist dem Autor bekannt, dass Bold for Delphi meist in kleinen bis mittelgroßen Enterpriseanwendungen verwendet wird und dabei hauptsächlich eine zweischichtige Client-Server Architektur mit einem Fat Client zum Einsatz kommt. Ein Fat Client bedeutet in diesem Zusammenhang, dass sich auf dem Server lediglich die Datenbank befindet und die Clients die gesamte Geschäftslogik enthalten. Das Framework kann und wird aber auch teilweise mittels SOAP (Simple Object Access Protocol) in Form von Webdiensten eingesetzt. Daneben lassen sich schnell kleine und lokale Anwendungen für den Einzelbenutzerbetrieb entwickeln.

Zur Installation: vorausgesetzt man nutzt bereits die IDE (Borland Developer Studio 2006), geht die Installation relativ schnell. Die letzte offizielle Version bietet kein

[52] Vgl. BoldSoft MDE AB, Bold for Delphi 2002
[53] S. <http://cc.embarcadero.com/item.aspx?id=23890>

Installationsprogramm mehr, die sogenannten Packages müssen daher manuell in der IDE registriert werden. Insgesamt dauert dieser Vorgang nicht mehr als 45 Minuten.

Bei der Installation werden 37 Demos mitgeliefert und eine Vorlage für ein Beispielprojekt. Eine Hilfe oder Dokumentation ist bei dieser letzten Version nicht mehr im Lieferumfang enthalten, sie lässt sich aber im Internet finden. Nahezu alle Komponenten sind dort dokumentiert. Es gibt außerdem ein Dokument mit einer Schritt-für-Schritt-Anleitung zu einem ersten Projekt, bei dem auch die Grundlagen der OCL und deren Verwendung in Bold erläutert werden. Innerhalb von ca. acht Stunden lassen sich so mit dem Beispielprojekt bereits die wichtigsten Grundlagen erlernen.

Viele Teile des Quellcodes sind einsehbar, einige Units liegen allerdings nur in kompilierter Form vor. Eine Lizenz mit Zugriff auf den vollständigen Quellcode kann mittlerweile aufgrund der o. g. Firmenübernahme nicht mehr erworben werden. Die letzte offizielle Version kann kostenlos als Beigabe zur IDE heruntergeladen werden. Das Framework kann insgesamt jedoch nicht als Open Source bezeichnet werden.

4.1.2 MDriven

Das MDriven Framework (ehemals ECO) ist der Nachfolger des ebenfalls hier vorgestellten Bold for Delphi. Es wurde für das .Net-Framework von Microsoft neu entwickelt, übernimmt dabei allerdings viele der Konzepte aus Bold. Es sind Code-Generatoren für C# und VB.Net enthalten.[54] Der Fokus für diese Arbeit liegt aber auf C#. Verglichen wird die z. Z. aktuelle Version 7.0.0.7904 in der IDE Microsoft Visual Studio Enterprise 2015. Die Sprache C# ist grundsätzlich plattformunabhängig, da sie in einen Zwischencode kompiliert wird, der wiederum von einer virtuellen Maschine auf dem Zielsystem ausgeführt wird. Das Framework zielt jedoch auf einen Einsatz in Windowsumgebungen ab.

MDriven ist ein komplexes Framework mit einem umfangreichen Funktionsumfang. Im Kern steht ein eigener Modelleditor (MDriven Designer), der neben der Struktur der Geschäftsmodelle in Form von Klassendiagrammen auch Zustandsautomaten modellieren kann. Darüber hinaus kann dort ein sog. ViewModel mit Unterstützung für mehrere Benutzeroberflächen (WPF, ASP.Net, Windows Form, …) entworfen

[54] Vgl. CapableObjects AB, MDriven Framework – Model driven framework formerly known as ECO | CapableObjects o. J.

werden. Der O/R-Mapper unterstützt ebenso unterschiedliche SQL-Datenbanksysteme.

Das Einsatzgebiet ist ebenso wie bei Bold for Delphi auf mittelgroße bis große Enterpriseanwendungen ausgelegt. Der Fokus verschiebt sich hier aber von klassischen Client-Server-Architekturen klar in Richtung Webdienste. Es wurde auch mehr Wert auf die Skalierbarkeit und einen gleichzeitigen Zugriff durch viele Benutzer gelegt. Trotzdem eignet sich das Framework nicht für Web 2.0-Anwendungen mit großen Datenmengen, da keine NoSQL-Datenbanksysteme unterstützt werden.

Die Installation[55] erfolgt schnell und automatisch per Installationsprogramm. Voraussetzung ist Visual Studio 2012-2015. Die Packages können – ebenso wie weitere speziellere Ressourcen – zusätzlich über die NuGet Paketverwaltung abgerufen werden. Danach hat man die Möglichkeit seinen Lizenzschlüssel einzugeben. Insgesamt dauert die Installation nicht mehr als 30 Minuten.

Mitgeliefert werden 2 Projektvorlagen sowie 38 Beispielprojekte, welche die einzelnen Konzepte vorstellen. Der zugängliche Quellcode ist überwiegend dokumentiert. Auf der Webseite gibt es weiterführende Dokumentationen: neben einem Schritt für Schritt Einstieg in die UML-Modellierung mit dem MDriven Designer, wird zurzeit ein MDriven Buch geschrieben, welches nach und nach alle wichtigen Konzepte des Frameworks beschreiben soll.[56] Es gibt außerdem eine noch aktive, wenn auch kleine Community und einen aktiven Support. Der Einstieg gelingt so relativ schnell. Innerhalb von zehn bis zwölf Stunden lässt sich auch hier eine Testanwendung mit im Modell entworfener Benutzeroberfläche erstellen.

MDriven ist kostenpflichtig, kann jedoch für kleinere Modelle von bis zu 50 Klassen kostenlos genutzt werden. Der Quellcode ist größtenteils nicht frei zugänglich. Die Kosten belaufen sich auf EUR 980,00 für einen Entwickler; der Zugriff auf den vollständigen Quellcode kann für ein Jahr für EUR 1.800,00 beantragt werden (Stand Mai 2016). Mit MDriven Turnkey steht außerdem ein SaaS-Produkt für Internetanwendungen zur Verfügung;[57] monatlich werden dann EUR 28,43 fällig.

---

[55] S. <http://www.new.capableobjects.com/downloads/>
[56] Vgl. CapableObjects AB, MDriven – the book | CapableObjects o. J.
[57] Vgl. CapableObjects AB, MDriven Turnkey | CapableObjects o. J.

4.1.3 Texo

Texo gehört zu den Eclipse Modeling Framework Technologies (EMFT) und setzt damit auf dem Eclipse Modeling Framework (EMF) auf. Das EMF ist die Modellierungslösung der Eclipse Foundation. Es besitzt ein eigenes Metamodell und ist gut in die Eclipse IDE integriert. Das EMF bietet neben einem Editor eine schablonenbasierte Quellcode-Generierung mit dem Fokus auf Java.[58] Java ist ebenso wie C# eine plattformunabhängige Sprache, die erst auf dem Zielsystem in nativen Maschinencode übersetzt wird. Verglichen werden die z. Z. aktuelle Version von Texo (0.9) und dem EMF (2.10) in der Eclipse Mars IDE (4.5.2).

Das Texo Framework[59] bietet – nach entsprechender Erweiterung der Modelle – neben der Generierung des Java-Quellcodes auch die Generierung des O/R-Mappings nach dem JPA Standard. Es kann dadurch theoretisch mit verschiedenen O/R-Mappern und vielen Datenbanksystemen genutzt werden. Für diesen Vergleich wird jedoch nur EclipseLink eingesetzt. Weiterhin ermöglicht das Framework, während der Laufzeit des Systems auf das Metamodell zuzugreifen. Im Gegensatz zum Vorgänger Teneo ist Texo auf den Einsatz in Webdiensten ausgelegt. Damit kann es auch für große Enterprise-Anwendungen mit vielen Benutzerzugriffen genutzt werden und ist entsprechend skalierbar.

Texo kann als Eclipse Plug-In über eine manuell anzugebende Quelle installiert werden.[60] Die wichtigste Abhängigkeit ist das EMF, das ebenfalls installiert sein muss. Ebenso kann Texo über das Maven Repository bezogen werden. Beispielprojekte befinden sich auf GitHub und müssen von dort gesondert heruntergeladen werden. Insgesamt kann die Installation und Einrichtung zwei bis drei Stunden dauern.

Die zuvor erwähnten Beispielprojekte sind in ihrer Anzahl überschaubar – im Wesentlichen handelt es sich um ein Beispielprojekt mit vier Modellen. Dazu gibt es eine Schritt-für-Schritt-Anleitung, um sie auf einem Tomcat 7 Application Server auszuführen. Im eigenen Wiki werden außerdem die wichtigsten Konzepte erläutert, was einen guten Überblick ermöglicht. Die Entwicklergemeinde scheint subjektiv betrachtet nicht besonders groß oder aktiv zu sein, aber sie ist vorhanden. Insgesamt verlangt das Framework eine gewisse Eigeninitiative, bevor man als Anwender das

[58] Vgl. Eclipse Foundation, Eclipse Modeling Project o. J.
[59] Vgl. dazu im Folgenden Eclipse Foundation, Texo - Eclipsepedia 2015
[60] Vgl. Eclipse Foundation, Texo/Download and Install - Eclipsepedia 2013

Framework gut versteht und selbstständig eigene Anwendungen erstellen kann. Voraussetzung sind bereits bestehende Kenntnisse im EMF. Bis zur ersten eigenen Anwendung können so gut 24 Stunden vergehen.

Texo ist als Open Source unter der Eclipse Public License (EPL) kostenlos verfügbar.

4.2 Detaillierter Funktionsumfang

Bei diesem Vergleichskriterium wird ein genauer Blick auf den Funktionsumfang geworfen. Insbesondere geht es in diesem Kapitel darum, die Gemeinsamkeiten und Alleinstellungsmerkmale der verschiedenen O/R-Mapping-Frameworks zu ermitteln und aufzuzeigen, welche Funktionen überhaupt mit diesen Frameworks möglich sind.

Dazu gehören zum einen die Funktionen des reinen O/R-Mappings i. e. S., wie sie bereits in Kapitel 2.2.3 beschrieben wurden:

- CRUD-Operationen
- Abfragen
- Navigation und Eager-/Lazy-Loading
- Transaktionskontrolle
- Cache
- Mehrbenutzer-Zugriff & Multitenancy
- Locking
- Versionierung

Außerdem wäre es interessant zu wissen, wie die O/R-Mapper das in Kapitel 2.2.2.3 beschriebene Problem der Objektidentität lösen.

Mittlerweile hat es sich bei den meisten O/R-Mappern durchgesetzt, das Inversion of Control (IoC) Entwurfsmuster anzuwenden. Im Falle des O/R-Mappings geht es darum, einfache Objekte (in Java sog. POJOs) zu verwenden, damit man diese ohne Abhängigkeiten zum O/R-Mapping zusätzlich in anderen Szenarien, z. B. in Unittests, einsetzen kann. Auch wenn diese Funktion nichts mit dem O/R-Mapping selbst zu tun hat, wäre eine Umsetzung durch den O/R-Mapper trotzdem wünschenswert, da das Entwurfsmuster für einen leichter wiederverwendbaren Quellcode sorgt. Die Umsetzung dieses Prinzips wird daher ebenso geprüft.

Zum anderen sollen bei diesem Vergleichskriterium Funktionen untersucht werden, wie sie speziell bei modellgetriebenen O/R-Mappern vorzufinden sind (s. Kapitel 2.3):

- Generierung des O/R-Mappings aus dem Modell heraus
- Abweichende Mapping Einstellungen im Modell definieren können
- Database Evolution
- Modellvalidierung
- Zugriff auf das Metamodell zur Laufzeit

Schließlich gibt es noch Funktionen auf der Ebene des Frameworks, die ebenso wie der O/R-Mapper das Modell als Basis nutzen, mit dem O/R-Mapping selbst aber nichts zu tun haben. Dazu gehört zum Beispiel die Generierung von Zustandsautomaten. Im Fokus stehen jedoch die o. g. Funktionen, die mit dem O/R-Mapping in Verbindung stehen. Alles darüber hinaus dient nur dem Überblick und der Einschätzung des Umfangs bzw. der Komplexität des Frameworks und begründet damit die Angabe in Kapitel 4.1.

Nicht betrachtet werden sollen hingegen Funktionen, die weder dem modellgetriebenen Ansatz des MDSD folgen, noch mit dem O/R-Mapping in Verbindung stehen, da sie nicht Thema dieser Arbeit sind.

4.2.1 Bold for Delphi

Die Abdeckung der Funktionen des reinen O/R-Mappings zeigt die folgende Tabelle:

Tabelle 2: O/R-Mapping-Funktionen von Bold for Delphi

Funktion	Umsetzung
CRUD	Alle CRUD-Funktionen sind einfach und intuitiv möglich.
Abfragen	Es sind beliebige typisierte Abfragen mittels OCL möglich, dank OCL2SQL auch performant direkt auf der Datenbank, ohne dabei nicht benötigte Objekte zu laden. Ebenso können native SQL-Abfragen abgesetzt werden.
Navigation	Die Navigation über Rollen ist ohne Zutun des Entwicklers möglich, die Daten werden automatisch geladen. Ist die Navigation im Voraus bekannt, können die Daten performant vorgeladen werden. Lazy- und Eager Loading werden gleichermaßen unterstützt. Bold arbeitet mit sog.

	Locatoren, die unabhängig vom Objekt geladen werden können. Die Objektdaten können bei Bedarf später geladen werden, während die Locator z. B. schon in Assoziationen verwendet werden können.
Transaktionen	Einfache und geschachtelte (nur geschlossene) Transaktionen sind möglich. Darüber hinaus gibt es im Speicher auch die Möglichkeit, Änderungen zusammenzufassen und ggf. wieder zu verwerfen (Undo-/Redo-Mechanismus)
Caching	Einmal geladene Daten bleiben im sog. Object Space bestehen, bis sie manuell entladen werden oder der Object Space zerstört wird. Ein erneutes Laden ist ebenso möglich.
Mehrbenutzer-Zugriff & Multitenancy	Ist möglich mit einem Fat Client: Jeder Client besitzt einen eigenen Object Space; für die Synchronisierung zwischen den Clients sowie das Locking sind Erweiterungen erforderlich. Ein einzelner Object Space ist nicht threadsicher.
Locking	Das Locking über alle Clients ist nur mit Erweiterungen möglich. Dabei wird grundsätzlich sowohl optimistisches als auch pessimistisches Locking unterstützt.
Versionierung	Die Versionierung von Objekten wird mittels Erweiterung rudimentär unterstützt.
Identität	Die Identität wird über eine fortlaufende Bold-ID für alle Objekte bestimmt, welche in einer speziellen Tabelle hochgezählt wird. Jedes Objekt ist damit eindeutig über diese ID identifizierbar.
Inversion of Control	Das Prinzip wird nicht eingehalten. Es gibt strikte Vorgaben für einen gemeinsamen Vorfahren und den Aufbau der Klassen.

Einen Blick auf den integrierten Modelleditor von Bold bietet die folgende Abbildung 5. Eine graphische Visualisierung ist damit allerdings nicht möglich.

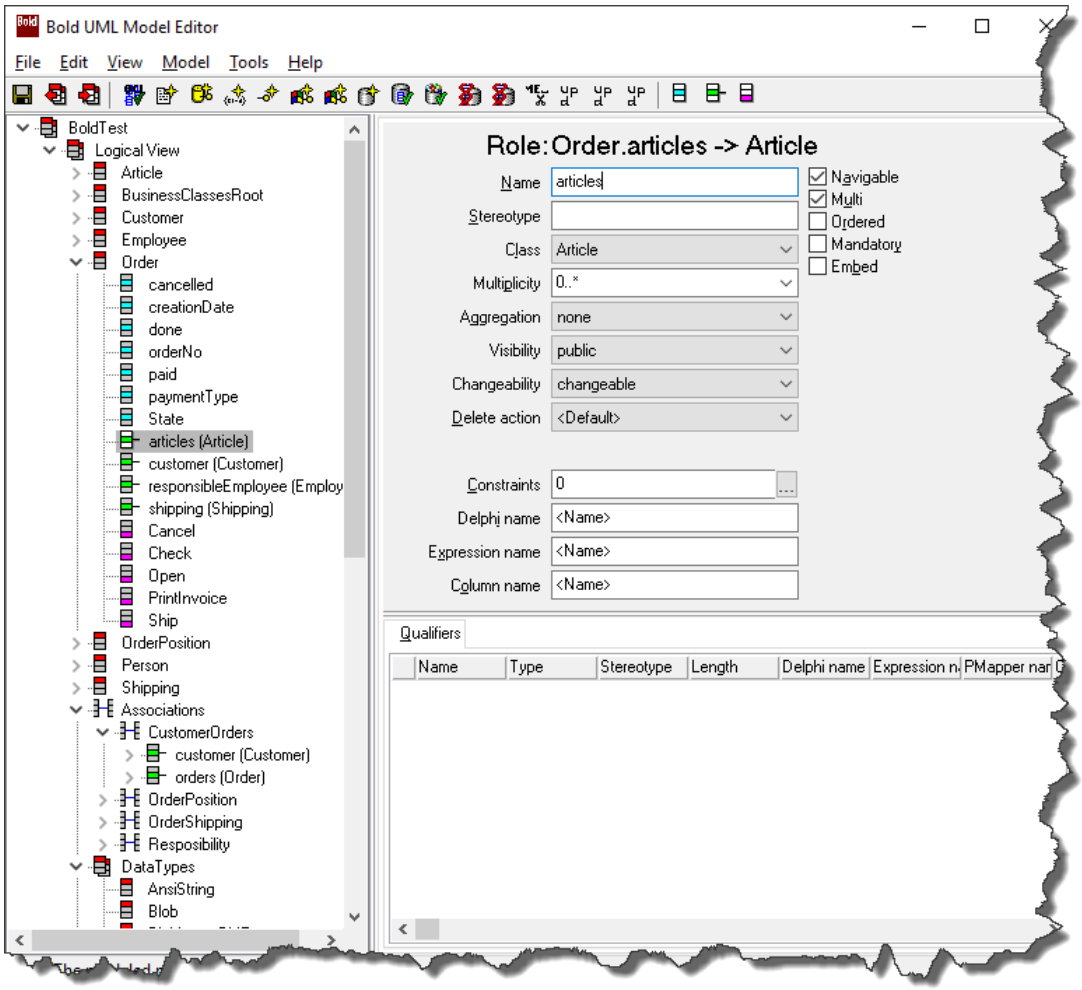


Abbildung 5: Bold UML Model Editor

Der Quellcode wie auch das O/R-Mapping werden vollständig aus dem Modell heraus generiert. Dabei wird der vom Entwickler manuell geschriebene Quellcode nicht wieder überschrieben. Es können über UML hinausgehende Einstellungen für das O/R-Mapping mittels sog. „tagged values“ (Schlüssel-Wert-Paaren[61]) im Modell angegeben werden. Beispiele sind die Angaben zum Table Mapping oder Locking-Verhalten, abweichende Tabellen-/Spaltennamen oder die Kennzeichnung von transienten Elementen.

Außerdem bietet Bold for Delphi die Möglichkeit, eigene Persistence Mapper für beliebige Attribute zu definieren, welche dann auch komplexe Datentypen über

[61] Vgl. Grose, Doney und Brodsky, Mastering XMI 2002, S. 47

mehrere Spalten in der Datenbanktabelle verteilen können. In einem gewissen Maße lassen sich so z. B. mehrsprachige Anwendungen erzeugen, indem für ein Attribut Zeichenketten in mehreren Sprachen in der Datenbank abgelegt werden.

Weiterhin unterstützt das Framework bis zu einem gewissen Grad die Database Evolution. Das heißt nach Modelländerungen kann automatisch ein SQL-Skript generiert werden, welches versucht, die Änderungen ohne Datenverlust auf das Datenbankschema zu übertragen. Dies wird dadurch erreicht, in dem die Angaben zum Modell bzw. O/R-Mapping ebenfalls in der Datenbank abgelegt werden und so Änderungen bei neuen Modellversionen abgeglichen werden können. Auf diese Weise können z. B. geänderte Datentypen oder in der Vererbungshierarchie verschobene Klassen oder Attribute erkannt werden.

Neben dem O/R-Mapping bietet das Framework mit Hilfe der OCL verschiedene weiterführende Funktionen auf Grundlage des Modells an. Neben der Angabe von Einschränkungen, für die OCL eigentlich gedacht ist, lassen sich überall beliebige Abfragen auf den Modellelementen durchführen. So können u. a. transiente Attribute und Assoziationen berechnet sowie GUI-Steuerelemente mit dem Modell verknüpft werden. Zusätzlich bietet das Modell einen Subscription-Mechanismus, mit dem man auf Änderungen an den Daten (auch über einen OCL) reagieren kann. Die Steuerelemente zeigen so immer den aktuellen Inhalt an.

Abschließend gibt es verschiedene Validierungsmöglichkeiten, die z. T. vor der Quellcode-Generierung erfüllt sein müssen. Neben der Bedingung, dass keine reservierten Wörter verwendet werden dürfen, ermittelt die Validierung z. B. doppelte Bezeichner oder fehlerhaft definierte Assoziationen. Außerdem können alle verwendeten OCL-Ausdrücke auf eine korrekte Schreibweise hin überprüft werden. Zur Laufzeit des Systems kann auf das komplette Metamodell zugriffen werden.

Das Verhalten, z. B. in Form von Zustandsautomaten, kann nicht aus dem Modell heraus generiert werden.

4.2.2 MDriven

Der Funktionsumfang wurde gegenüber dem Vorgänger Bold erweitert. Viele Funktionalitäten, die sich vorher nur mit Erweiterungen realisieren ließen, sind nun fester Bestandteil des Frameworks. Zunächst zeigt Tabelle 3 wieder die Abdeckung der Funktionen des reinen O/R-Mappings:

Tabelle 3: O/R-Mapping-Funktionen von MDriven

Funktion	Umsetzung
CRUD	Alle CRUD-Funktionen sind einfach und intuitiv möglich.
Abfragen	Es sind beliebige typisierte Abfragen mittels OCL möglich, dank OCL2SQL auch performant direkt auf der Datenbank. Ebenso können native SQL-Abfragen abgesetzt werden.
Navigation	Die Navigation über Rollen ist ohne Zutun des Entwicklers möglich, die Daten werden automatisch geladen. Ist die Navigation im Voraus bekannt, können die Daten performant vorgeladen werden. Lazy- und Eager Loading werden gleichermaßen unterstützt.
Transaktionen	Einfache und geschachtelte (nur geschlossene) Transaktionen sind möglich. Darüber hinaus gibt es im Speicher auch die Möglichkeit, Änderungen zusammenzufassen und ggf. wieder zu verwerfen (Undo-/Redo-Mechanismus)
Caching	Einmal geladene Daten bleiben im sog. Object Space bestehen, bis sie manuell entladen werden oder der Object Space zerstört wird. Ein erneutes Laden ist ebenso möglich.
Mehrbenutzer-Zugriff & Multitenancy	Gleichzeitiger Zugriff mehrerer Benutzer wird unterstützt. Darüber hinaus lassen sich auch Multitenancy-Architekturen realisieren (Kennzeichnung einzelner Klassen ist möglich).
Locking	Es wird sowohl optimistisches als auch pessimistisches Locking unterstützt.
Versionierung	Die Versionierung von Objekten wird unterstützt.

Identität	Die Identität wird über eine fortlaufende ECO-ID für alle Objekte bestimmt, welche in einer speziellen Tabelle hochgezählt wird. Jedes Objekt ist damit eindeutig über diese ID identifizierbar.
Inversion of Control	Das Prinzip wird nicht eingehalten. Im Gegensatz zu Bold gibt es zwar keinen gemeinsamen Vorfahren, jedoch muss ein Interface implementiert werden und der Aufbau der Klassen ist weiterhin vorgegeben.

Das O/R-Mapping wird vollständig automatisiert aus dem Modell heraus generiert. Dabei können über UML hinausgehende Einstellungen für das O/R-Mapping ebenfalls mittels tagged values im Modell angegeben werden (wie im Vorgänger Bold).

Das Thema Database Evolution wurde gegenüber Bold weiter verbessert. Wie beim Vorgänger werden hier alle Informationen über das Modell und das O/R-Mapping mit in der Datenbank abgespeichert, damit Änderungen erkannt werden können.

MDriven bietet außerdem die Ausführung von Modellen für das Prototyping. Diese Funktion kann direkt aus dem Modelldesigner heraus angestoßen werden, ohne zuvor den Quellcode generiert oder eine Datenbank angelegt zu haben. Das Prototyping kann damit vollständig im Speicher ausgeführt werden. Über automatisch generierte Formulare kann dann mit den Objekten des Systems interagiert werden.

Zu den erweiterten Funktionen, die dem modellgetriebenen Ansatz folgen, bietet MDriven die Möglichkeit, sog. ViewModels zu designen, die dann für verschiedene Benutzeroberflächen die entsprechenden Formulare generieren. Dazu gehören neben der Auswahl und Anordnung der Inhalte auch die zur Verfügung stehenden Aktionen, welche dann direkt im Modell mithilfe der eigenen Eco Action Language (EAL, ähnlich den OCL-Ausdrücken) beschrieben werden können. Abbildung 6 zeigt beispielhaft den ViewModel-Editor für ein Order-Objekt aus dem Beispielmodell.

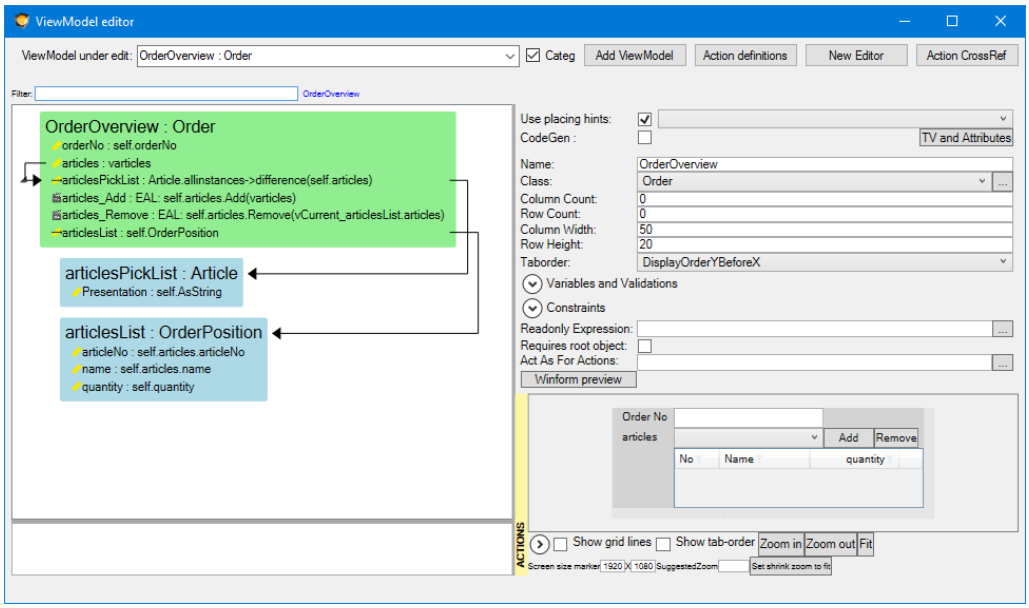


Abbildung 6: MDriven ViewModel-Editor

Außerdem können Zustandsautomaten modelliert werden, wie in Abbildung 7 zu sehen ist. Die Zustandsübergänge können durch Operationen aus dem Modell getriggert werden und es können u. a. Guards und Effekte mithilfe von EAL und OCL angegeben werden. Die Zustandsautomaten, ebenso wie die ViewModels, sind Alleinstellungsmerkmale dieses Frameworks.

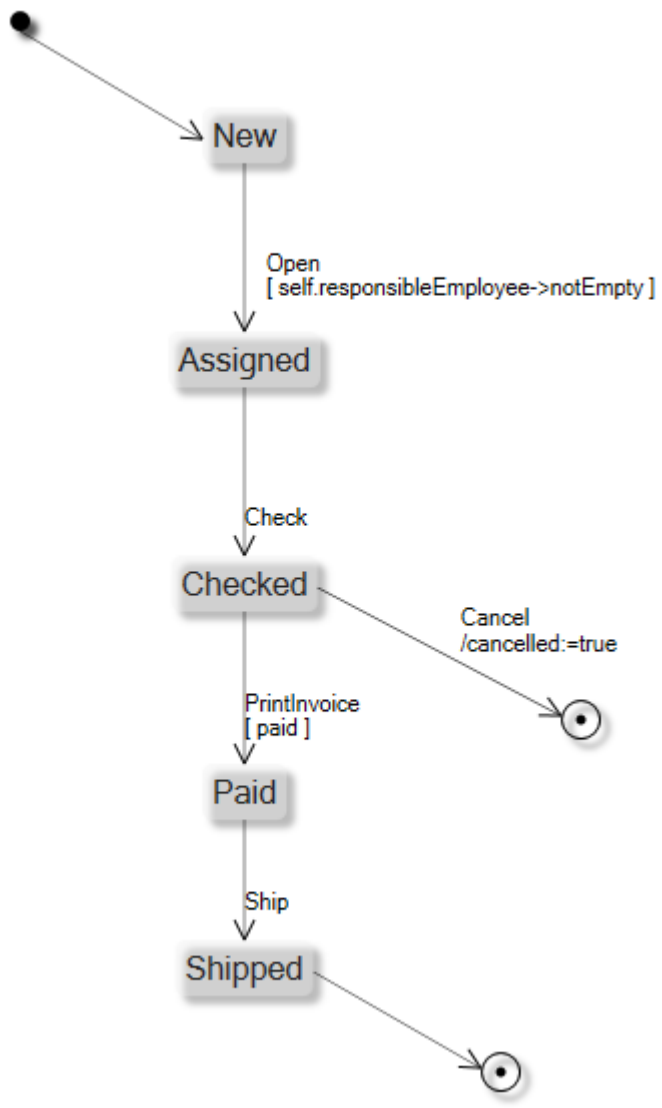


Abbildung 7: MDriven Zustandsautomat

Alle genannten Modellelemente (inkl. der ViewModels und Zustandsautomaten) sind miteinander verzahnt und können abschließend auch zusammen validiert werden. Syntaktische Fehler oder fehlerhafte OCL-Ausdrücke können so einfach gefunden werden. Ebenso werden wie beim Vorgänger fehlerhafte Mapping-Einstellungen erkannt. Im Vergleich erreicht kein anderes Framework eine so starke Integration aller Komponenten.

4.2.3 Texo

Als Erweiterung des EMF, werden hier zunächst die für Texo relevanten Funktionen und Konzepte des Modellierungsframeworks erläutert, bevor dann auf den Funktionsumfang von Texo im Detail eingegangen wird.

Das EMF[62] besteht im Kern aus dem ECore-Metamodell – einem Domänenmodell, welches im Wesentlichen eine Teilmenge der UML-Klassendiagramme abbildet und speziell auf Java zugeschnitten ist. Aus einem ECore-Modell lassen sich dann beliebige Artefakte generieren, u. a. Java-Quellcode. Die Generierung erfolgt i. d. R. durch die Template Engine JET (Java Emitter Templates). Hierbei ist es ebenso möglich, manuelle Änderungen im generierten Quellcode vorzunehmen, die bei einer erneuten Generierung nicht überschrieben werden. Ebenfalls bietet das Framework Zugriff auf die Metainformationen zur Laufzeit sowie eine XML-Serialisierung.

Texo nutzt EMF in erster Linie als Quelle für die Metadaten und zum Zugriff auf die Modellinformationen zur Laufzeit. Die Generierung der Artefakte erfolgt hingegen eigenständig mit speziellen Templates für die Template Engine XPand sowie mit XTend. Es gibt einige EMF-Funktionalitäten, die nicht von Texo unterstützt werden. So kann Texo beispielsweise nur mit normalen Collections für Assoziationen umgehen. Ebenso gibt es keine Unterstützung für den Benachrichtigungsmechanismus bei Änderungen im Modell, da dies für Webanwendungen als Zielumgebung nicht geeignet ist. Auch auf Constraints in Form von OCL-Ausdrücken muss verzichtet werden, obwohl diese Sprache von EMF mittels Erweiterung ebenfalls unterstützt wird.

Die Quellcode-Generierung kann in Texo über ein sog. „Annotations Model“ gesteuert werden, in welchem spezifische Einstellungen definiert werden. Dazu gehört z. B. von welcher Klasse abgeleitet wird, wie Assoziationen behandelt werden oder wie ein Element mittels javadoc dokumentiert werden soll. Spezielle Mapping-Einstellungen können ebenfalls dort angegeben werden. Das ECore-Modell wird dann mit diesem Modell zusammengeführt (s. M2M-Transformation). Neben dem ECore-Modell wird darüber hinaus die schlanke Erweiterung XCore unterstützt. Für das O/R-Mapping kann JPA2-konform entweder annotierter Java-Quellcode oder eine orm.xml-Datei generiert werden. Damit lassen sich grundsätzlich alle JPA2-Provider verwenden; bevorzugt wird allerdings die Referenzimplementierung EclipseLink.

Mit EclipseLink steht ein vollwertiger O/R-Mapper zur Verfügung, mit dem sich alle wichtigen Funktionen des O/R-Mappings realisieren lassen:

---

[62] Vgl. dazu im Folgenden Koegel und Helming, EMF Tutorial « EclipseSource Blog 2016 und Eclipse Foundation, Texo/Texo and EMF - Eclipsepedia 2010

Tabelle 4: O/R-Mapping-Funktionen von Texo/EclipseLink

Funktion	Umsetzung
CRUD	Alle CRUD-Funktionen sind einfach und intuitiv möglich.
Abfragen	Die Java Persistence Query Language (JPQL), welche an den bekannten SQL-Syntax angelehnt ist, ermöglicht beliebige typisierte Abfragen auf der Datenbank und dem Cache (also auf im Speicher geladenen Daten).
Navigation	Die Navigation über Rollen ist ohne Zutun des Entwicklers möglich, die Daten werden automatisch geladen. Ist die Navigation im Voraus bekannt, können die Daten über einen sog. Batch performant vorgeladen werden (dies ist nicht mit allen JPA-Providern möglich, aber z. B. mit EclipseLink). Lazy- und Eager Loading werden gleichermaßen unterstützt.
Transaktionen	Einfache und geschachtelte (nur geschlossene) sowie parallele Transaktionen sind möglich.[63] Ein fertiger Undo-/Redo-Mechanismus wie bei den anderen Frameworks konnte nicht gefunden werden.
Caching	Einmal geladene Daten können in einem Cache vorgehalten werden. Es gibt auch einen Zwischenspeicher für Abfragen.
Mehrbenutzer-Zugriff & Multitenancy	Gleichzeitiger Zugriff mehrerer Benutzer wird unterstützt. Darüber hinaus lassen sich auch Multitenancy-Architekturen realisieren (Kennzeichnung einzelner Klassen ist möglich).
Locking	Es wird sowohl optimistisches als auch pessimistisches Locking unterstützt.
Versionierung	EclipseLink kann grundsätzlich Historienstände vorhalten, Texo bietet jedoch im Gegensatz zum Vorgänger Teneo keine native Schnittstelle mehr für diese Funktion.
Identität	Die Identität wird über eine ID-Spalte mit einem generierten Wert erzielt (Standard: JPA-GenerationType

[63] Eclipse Foundation, Introduction to EclipseLink Transactions (ELUG) - Eclipsepedia 2012

	AUTO). Es kann hier also z. B. herstellerabhängig auch eine Sequence verwendet werden.
Inversion of Control	Bei Bedarf können POJOs generiert werden, die Ableitung von BaseEObjectImpl oder die Implementierung von EObject wie in EMF ist nicht erforderlich.

EclipseLink bietet lediglich eine einfache Form der Database Evolution, bei der allerdings nicht die Informationen aus dem Modell zur Hilfe genommen werden. Nur das Hinzufügen neuer Tabellen oder Spalten gelingt dabei automatisiert und ohne Datenverlust.

Weiterhin unterstützt Texo die Generierung von XML und JSON Webdiensten. Auch DAO-Objekte (Data Access Object) können generiert und genutzt werden. Eine weitere interessante Funktion ist die modellgetriebene Generierung von zufälligen Testdaten. Dies ist insbesondere in frühen Stadien der Entwicklung beim Thema Prototyping nützlich.

4.3 Umsetzung der Model Driven Architecture

Bei diesem Vergleichskriterium geht es darum, zu ermitteln, inwiefern das modellgetriebene O/R-Mapping-Framework den Ansatz der Model Driven Architecture umsetzt.

Dazu kann ganz allgemein untersucht werden, welche Artefakte (z. B. Quellcode, O/R-Mapping, GUI, Test, Dokumentation) sich aus dem Modell heraus generieren lassen. Beim Quellcode ist es zudem wichtig, welche der Modellelemente sich automatisiert in Quellcode übersetzen lassen. Beispiele hierfür wären die Klassenhierarchie, Zustandsautomaten oder Aktivitäten. Einen genaueren Blick auf die UML-Funktionen, welche zur Generierung genutzt werden können, bietet hier aber erst das nachfolgende Kapitel 4.4.

Außerdem wird hier untersucht, ob die unterschiedlichen Abstraktionsstufen der Modelle (CIM, PIM und PSM) genutzt werden oder ob vielleicht auf eine oder sogar zwei dieser Stufen verzichtet wird.

4.3.1 Bold for Delphi

Im Vordergrund von Bold for Delphi steht neben der Generierung des O/R-Mappings die Quellcode-Generierung für Delphi. Umgesetzt werden alle strukturellen Modellelemente, also die Klassenhierarchie. Die Methodenrümpfe lassen sich im Modell ebenfalls definieren; die Logik muss allerdings vom Entwickler im generierten Quellcode selbst geschrieben werden. Eine Unterstützung für Zustandsautomaten oder Aktivitätsdiagramme liegt damit nicht vor.

Bold bietet eine Reihe von GUI-Komponenten und viele Hilfsmittel zum Erstellen von Windows-Forms-Benutzeroberflächen. Dazu gehören auch die sog. Auto Forms, mit deren Hilfe sich alle Informationen zu einem Modellelement automatisiert darstellen lassen. Allerdings kann man auf deren Aussehen keinen Einfluss nehmen. Ein modellbasiertes Entwerfen der Benutzeroberfläche ist daher nicht möglich. Ebenso können keine Tests oder eine Dokumentation aus dem Modell heraus generiert werden. Eine entsprechende Erweiterung würde sich aber realisieren lassen. Bold bietet außerdem eine integrierte Modellvalidierung.

Grundsätzlich kann Bold mit UML-Modellen umgehen und diese importieren. Die Modellinformationen lassen sich allerdings nicht mit den Bold-spezifischen Informationen anreichern. Entweder sind sie daher bereits im vorhergehenden Modellierungswerkzeug anzugeben (explizit unterstützt werden beispielsweise Rational Rose und Model Maker) oder man arbeitet vollständig mit Bold. Das Konzept der unterschiedlichen Abstraktionsstufen wird von Bold daher nicht umgesetzt, es gibt lediglich das PSM.

Zusammenfassend beherrscht Bold lediglich eine der Kerndisziplinen der MDA – die Generierung einiger Artefakte – und gliedert sich damit nicht gut in den gesamten Entwicklungszyklus ein. Eine reproduzierbare Modell-Transformation gelingt nicht. Bold kann praktisch nur als Startpunkt des MDA-Ansatzes genutzt werden und dann leider nur auf der geringsten Abstraktionsstufe.

4.3.2 MDriven

MDriven ist ebenso wie sein Vorgänger Bold nicht auf Plattformunabhängigkeit ausgelegt. Die Generierung des Quellcodes fokussiert sich auf das .Net-Framework. Im Gegensatz zum Vorgänger wurde die Anzahl der generierbaren Artefakte aber deutlich erhöht. Neben der Klassenhierarchie lassen sich bei MDriven auch

Zustandsautomaten im Modell modellieren. Der Entwurf der Benutzeroberfläche ist mit Hilfe sog. ViewModels direkt innerhalb des Modells möglich. Damit lassen sich verschiedene Benutzerschnittstellen je nach Anwendungsszenario automatisch generieren. Ebenso lässt sich jetzt eine Dokumentation im Modell hinterlegen, welche bei der Generierung als Quellcode-Kommentar eingefügt wird. Eine Modellvalidierung inkl. der Validierung der ViewModels ist gleichfalls möglich.

Aus Sicht des MDA-Ansatzes hat sich damit aber im Vergleich zum Vorgänger in der Summe nichts verbessert: es bleibt bei einem plattformabhängigen Modell, welches nicht automatisiert aus einem plattformunabhängigen hervorgehen kann. Lediglich die Generierungsmöglichkeiten wurden erweitert. Zudem wurde die Unterstützung für das XMI-Format entfernt, sodass keine Modelle aus Fremdanwendungen importiert werden können. Nur der Import/Export von Bold-Modellen aus dem Vorgänger ist möglich. Es handelt sich zwar um MDSD, mit dem standardisierten MDA-Ansatz der OMG hat dies jedoch nicht mehr viel gemein.

4.3.3 Texo

Auch bei Texo steht die Generierung des Quellcodes und des O/R-Mappings im Vordergrund. Die Generierung kann durch die Template Engine vom Entwickler erweitert werden. Da EMF nur strukturelle Modellelemente enthält, kann kein Verhalten generiert werden.

Die Benutzeroberfläche kann bei Texo zwar rudimentär in Zusammenarbeit mit dem Framework gestaltet werden, gehört jedoch nicht zum Modell. Die Fähigkeit, Artefakte aus dem Modell heraus zu generieren, entspricht damit ungefähr dem Bold-Framework.

Wesentlich besser ist bei Texo allerdings die Integration in den Entwicklungszyklus gelungen. Dank EMF können Modelle im XMI-Format als Ausgangspunkt verwendet werden, welche dann durch das bereits beschriebene sog. „Annotations Model“ um die Texo-spezifischen Informationen erweitert werden können. Texo lässt sich damit in einen vollständigen MDA-Zyklus, bestehend aus CIM, PIM und PSM, integrieren. Texo selbst nutzt zwei Abstraktionsstufen (ECore-Modell und annotiertes ECore-Modell), die beide im PSM-Bereich angesiedelt sind.

4.4 Nutzung von Standards

Die Nutzung von Standards ist gerade im Bereich der MDA ein wichtiges Vergleichskriterium. Umso mehr standardisiert ist, desto leichter ist der Austausch einzelner Komponenten, der Plattform oder gegebenenfalls des gesamten Frameworks. Dadurch verringert sich die Bindung an Komponenten oder Frameworks, was aus Sicht des Entwicklers von Vorteil ist. Einzig das Modell bleibt als Kern über den gesamten Lebenszyklus bestehen und bildet damit den eigentlichen Wert eines Softwaresystems. Deshalb soll die Modellierung auch in diesem Kapitel im Vordergrund stehen. Standards erlauben darüber hinaus den anwendungsunabhängigen Austausch von Daten – sowohl von den Metadaten als auch den eigentlichen Daten – und verringern damit die Anzahl der benötigten Schnittstellen.[64]

Als Standard für die Modellierung hat sich die Unified Modeling Language (UML) etabliert. Bei diesem Vergleichskriterium wird deshalb ein genauerer Blick auf die unterstützten UML-Funktionen geworfen. Hierzu gehört eine Aufzählung der Modellelemente bzw. Diagrammtypen, die für die Generierung der Artefakte genutzt werden können. Die Basis bieten hier in der Regel die Klassen bzw. das Klassendiagramm. Teilweise unterstützen die Frameworks auch die Abbildung von Zuständen in der Programmlogik. Idealerweise sollte so viel wie möglich mithilfe der Standard-UML-Elemente abgebildet werden. Umso eher ist das Modell wiederverwendbar und umso leichter lässt sich dann das Framework austauschen. Ist das Standard-UML aber nicht ausreichend, können beliebige Informationen mit den sog. tagged values oder auch mit Hilfe von Stereotypen angegeben werden. Noch besser eigenen sich hingegen spezielle UML-Profile. Diese Profile erweitern das Standard-UML-Profil um eigene, fest im Profil definierte Stereotypen, tagged values oder Einschränkungen; Profile sind dabei besser standardisiert als frei wählbare tagged values und ebenso sind Modelle gegen diese Profile prüfbar.[65] Schließlich ist die verwendete bzw. unterstützte UML-Version relevant (UML 1 oder UML 2.x).

Für den Datenaustausch und die Speicherung von Metadaten kommt z. B. XML in Frage. Bei der Extensible Markup Language (XML)[66] handelt es sich um ein

---

[64] Vgl. Grose, Doney und Brodsky, Mastering XMI 2002, S. 7 & 14
[65] Vgl. Pilone, UML 2.0 in a Nutshell 2005, S. 169, 171
[66] Vgl. dazu im Folgenden Grose, Doney und Brodsky, Mastering XMI 2002, S. 3, 5f

standardisiertes, universelles Dateiformat, mit dem sich beliebige strukturierte Daten speichern lassen. Da es anwendungs- und plattformunabhängig ist, findet es insbesondere im World Wide Web große Anwendung. Neben dem Inhalt kann auch die Art und Weise, wie dieser Inhalt in XML dargestellt wird, mittels einer DTD (DOCTYPE, Dokumenttypdefinition) oder XML-Schemas angegeben werden. Dies vereinfacht den Datenaustausch mit anderen Anwendungen und ermöglicht eine syntaktische Validierung der Daten.

Für das Modell selbst hat sich darauf aufbauend XML Metadata Interchange (XMI) als Standard etabliert und wird von vielen MDA-Werkzeugen unterstützt. XMI ermöglicht zum einen den flexiblen Austausch von objektorientierten Daten, zum anderen spezifiziert es, wie XML-Schemas oder DTDs für ein Modell erzeugt werden bzw. wie man umgekehrt das Modell aus diesen erhält.[67] Die OMG sieht im Zuge der MOF XMI als Standard zum Austausch von (Meta-)Modellen vor.[68] Ein modellgetriebenes O/R-Framework, welches auf UML zur Modellierung setzt bzw. von der MOF Gebrauch macht, sollte daher ebenfalls XMI verwenden oder Import-/Exportmöglichkeiten für dieses Format bieten.

Auch die Generatoren können in gewisser Weise standardisiert arbeiten, z. B. in Form von allgemein verwendbaren Template Engines.

Für die Datenbankabfragen auf relationalen Datenbanken wird meist die Structured Query Language (SQL) benutzt. Diese ist zwar standardisiert, wird aber leider nicht in jedem Datenbanksystem einheitlich oder vollständig unterstützt (s. Kapitel 2.2.1). Lediglich ANSI-SQL1 aus dem Jahre 1986 wird i. d. R. von allen relationalen Datenbanksystemen vollständig unterstützt, SQL2/SQL92 nur zum Teil. Neben dem Standard gibt es oft noch herstellerabhängige SQL-Dialekte oder spezielle Funktionen für die verschiedenen Datenbanksysteme. Dazu gehören z. B. die in Kapitel 2.2.2.3 beschriebenen unterschiedlichen Verfahren zur Erzeugung einer eindeutigen ID. Idealerweise sollten die O/R-Mapper also viele Datenbanksysteme unterstützen, indem zunächst nur vom SQL1-Standard Gebrauch gemacht wird. Wenn möglich sollten optional aber gleichermaßen die neueren SQL-Standards und die

[67] Vgl. Grose, Doney und Brodsky, Mastering XMI 2002, S. 3, 10ff
[68] Vgl. Petrasch und Meimberg, Model Driven Architecture 2006, S. 14

herstellerabhängigen Spezialisierungen unterstützt werden, da diese häufig eine bessere Performance ermöglichen.

SQL-Abfragen werden aber normalerweise nur intern vom O/R-Mapper gebildet und genutzt. Dem Entwickler stehen meistens an SQL angelehnte Abfragesprachen zur Verfügung, welche auf der Ebene der Objekte arbeiten. Bekannte Beispiele dieser Abfragesprachen sind:

- Object Query Language (OQL): standardisierte Abfragesprache für Objektdatenbanken
- LINQ (Language Integrated Query) aus dem Hause Microsoft
- JPQL (Java Persistence Query Language): objektorientierte Abfragesprache als Teil der JPA-Spezifikation

Die Abfragesprachen gehören, sofern sie nicht auch vom Modell genutzt werden, zum O/R-Mapper i. e. S. und stehen nicht im Fokus dieser Arbeit. Für den Vergleich ist es daher ausreichend, die unterstützten Sprachen zu benennen ohne weitere Analysen bzgl. der Leistungsfähigkeit zu betreiben.

Schließlich gibt es noch weitere Standards, die gerade für den modellgetriebenen Ansatz relevant sind. Hierzu gehört u. a. die Object Constraint Language (OCL). Mit dieser Sprache können Constraints – also Einschränkungen/Restriktionen – in einem objektorientierten Modell für verschiedene Modellelemente spezifiziert werden.[69] Mit der OCL lassen sich ebenfalls Abfragen auf der Datenmenge tätigen.

4.4.1 Bold for Delphi

Bold verwendet UML 1.3 als Basis für das Metamodell. Zum Austausch des Modells kann im XMI-Format 1.0 importiert und exportiert werden. Bold unterstützt lediglich die strukturellen Elemente aus dem Klassendiagramm. Dazu gehören

- Pakete (lediglich zur Strukturierung)
- Klassen
    - Attribute, inkl. berechnete
    - Operationen mit Parametern (nur Signatur, ohne Beschreibung)
    - Abstrakte Klassen

[69] Vgl. Warmer und Kleppe, The Object Constraint Language 2000, S. 1

    - Schnittstellen werden nicht unterstützt
    - Einfachvererbung
- Assoziationen zwischen Klassen mit genau zwei Assoziationsenden
    - Aggregationen und Kompositionen
    - Multiplizitäten, Navigierbarkeit, geordnete und berechnete Assoziationen sowie Qualifizierungen
- Einfache Aufzählungstypen und Datentypen
- Sichtbarkeiten

Bold verwendet zur Angabe spezieller Eigenschaften, die für das Framework und das O/R-Mapping relevant sind, lediglich tagged values. Diese können beliebig angepasst und erweitert werden.

Zudem können an vielen Modellelementen Constraints in Form von OCL-Ausdrücken definiert werden, welche zur Laufzeit auf ihre Erfüllung überprüft werden können. Die OCL kommt in Bold aber nicht nur zur Definition von Einschränkungen im Modell zum Einsatz, sondern sie dient zusätzlich als Abfragesprache. Dank OCL2SQL können viele Sprachelemente auch direkt in SQL-Befehle umgewandelt werden, um so performante Abfragen direkt auf der Datenbank zu ermöglichen. Darüber hinaus können OCL-Ausdrücke für berechnete Attribute und Assoziationen angegeben werden (andernfalls werden Methodenrümpfe für die manuelle Berechnung bei der Quellcode-Generierung erzeugt).

Die Generierung der Artefakte erfolgt als M2T-Transformation mit einer eigenen Template Engine. Die Vorlagen können allerdings angepasst und erweitert werden.

Bold kommt grundsätzlich mit den SQL1-Funktionen aus und kann optional mit einigen SQL2-Funktionen umgehen (z. B. den dort spezifizierten Join-Operationen). Neuere SQL-Standards werden hingegen nicht unterstützt. Die SQL-Datentypen können aber frei eingestellt werden, wodurch viele Datenbankhersteller unterstützt werden: Advantage, DBISAM, Informix, Interbase, Oracle, Paradox, Postgres und Microsoft SQLServer.

4.4.2 MDriven

Bei den UML-Elementen aus dem Klassendiagramm hat sich gegenüber dem Vorgänger Bold in Bezug auf den UML-Standard keine nennenswerte Änderung ergeben. Sie entsprechen damit der Aufzählung aus Kapitel 4.4.1.

Dafür bietet MDriven Unterstützung für Zustandsautomaten bzw. Zustandsdiagramme. Dabei können die folgenden UML-Elemente genutzt werden:

- Zustände (inkl. Anfangszustand und Endzustände)
    - Verhalten bei Ein- und Austritt
- Transitionen
    - Effekte, Wächter (Guards) und Trigger

Alle weiteren Modellelemente entsprechen nicht dem UML-Standard.

Auch MDriven kommt grundsätzlich mit dem SQL1-Standard aus, bietet gegenüber dem Vorgänger Bold aber Unterstützung für weitere SQL-Befehle (z. B. TOP und LEN). Zu den offiziell unterstützten SQL-Datenbanksystemen gehören: Blackfish, Firebird, Microsoft SQLServer, Mimer, MySQL, NexusDB, Oracle, SQLite und Sybase; weitere sind leicht hinzuzufügen.[70] Ebenso steht bei MDriven wieder OCL als Abfragesprache (inkl. OCL2SQL) zur Verfügung. Daneben bietet MDriven Unterstützung für LINQ, allerdings nur mit einer Teilmenge der Funktionen, die über OCL zur Verfügung stehen.

Wie schon im vorherigen Kapitel beschrieben, bietet MDriven keine Unterstützung mehr für den Import/Export von UML-Modellen via XMI, auch wenn das interne Metamodell auf UML basiert. Ebenso ist die Generierung der Artefakte nicht mehr leicht erweiterbar. Die neuen Funktionen (z. B. das ViewModel) entsprechen auch keinem bekannten Standard. Die Metadaten dazu sind schemafrei in XML-Dateien abgelegt. Insgesamt hat sich MDriven von der Standardisierung abgewandt, was die Bindung an dieses Produkt deutlich erhöht.

4.4.3 Texo

Dank EMF bietet Texo umfangreiche Möglichkeiten für den Metadatenaustausch. Zu den unterstützen UML/XMI-Formaten gehören UML1.4 sowie UML2.x und XMI ab

[70] Vgl. CapableObjects AB, MDriven Framework – Model driven framework formerly known as ECO | CapableObjects o. J.

der Version 2.0. Außerdem können Rose Modelle importiert werden. Zu den unterstützten UML-Elementen gehören:

- Pakete
- Klassen
    - Attribute, inkl. berechnete
    - Operationen mit Parametern (nur Signatur, ohne Beschreibung)
    - Abstrakte Klassen
    - Schnittstellen
    - Einfachvererbung
- Assoziationen zwischen Klassen mit genau zwei Assoziationsenden
    - Kompositionen, aber keine Aggregationen
    - Multiplizitäten, Navigierbarkeit, geordnete und berechnete Assoziationen sowie Qualifizierungen
- Aufzählungstypen und Datentypen
- Sichtbarkeiten

Texo arbeitet mit der Template Engine XPand bzw. XTend. Die Vorlagen können leicht erweitert werden und im Internet lassen sich viele Beispiele für die Verwendung der Template Engine finden, da sie sich in Verbindung mit EMF als Standard etabliert hat.

EclipseLink als JPA-Provider bietet eine umfangreiche SQL-Unterstützung an. Für viele Datenbankhersteller gibt es spezielle Erweiterungen, die herstellerabhängige Datenbank- und SQL-Funktionen nutzen. Dazu gehören in der aktuellen EclipseLink Version 2.6 folgende Datenbanksystem: Apache Derby, Attunity, dBASE, Firebird, H2, HSQL, IBM Cloudspace, IBM DB2, IBM Informix, Microsoft Access, Microsoft SQLServer, MySQL, Oracle, PointBase, PostgreSQL, SAP MaxDB, Sybase, Fujitsu Symfoware.[71]

Als objektorientierte Abfragesprache kann gemäß der JPA-Spezifikation die JPQL verwendet werden. Daneben gibt es noch die speziell auf EclipseLink zugeschnittenen sog. „EclipseLink expressions“.

[71] Vgl. Eclipse Foundation, Database Support | EclipseLink 2.6.x Understanding EclipseLink 2015

4.5 Modularisierbarkeit

Beim letzten Vergleichskriterium soll auf die Modularisierbarkeit näher eingegangen werden. Die Fähigkeit, eine Anwendung und damit ebenfalls das zugrundeliegende Modell in verschiedene Module aufteilen zu können, spielt insbesondere bei umfangreichen und komplexen Softwareprodukten eine wichtige Rolle. Kunden wünschen häufig, dass sie sich die gewünschten Module selbst zusammenstellen können.[72] Die Probleme, die insbesondere beim modellgetriebenen Ansatz auftreten können, liegen darin begründet, dass das O/R-Mapping-Framework die einzelnen Module kennen bzw. aus dem Modell auslesen muss, um diese dann in korrekter Weise u. a. bei der Quellcode-Generierung zu beachten.[73]

Es muss also untersucht werden, ob und in wie weit das Framework die Modularisierung unterstützt und wie komfortabel dies erreicht wird. Die Modularisierung kann sich dabei rein auf das Modell, möglicherweise aber auch auf die Datenbasis beziehen. Letzteres würde es ermöglichen, dass z. B. sicherheitskritische Daten besonders geschützt in einer anderen Datenbank liegen, obwohl sie im gleichen Modell wie die anderen Geschäftsdaten beschrieben werden und mit diesen in Beziehungen stehen können. Diese Aufteilung der Datenbasis ist selbst bei codegetriebenen O/R-Mappern eher schwierig umzusetzen und wird nur selten unterstützt.

4.5.1 Bold for Delphi

Bold for Delphi bietet keine Funktionen zur Modularisierung an. Es können zwar Pakete innerhalb eines Modelles gebildet werden, diese dienen allerdings lediglich der Übersichtlichkeit und Strukturierung der Klassen. Ebenso wenig können die Daten auf mehrere Datenbanken aufgeteilt werden. Auf diesem Umstand basiert auch die vorherige Arbeit des Autors, bei der ein Konzept vorgestellt wurde, um zumindest auf der Seite des Modells eine Modularisierung zu erlauben.[74] Das Ergebnis war, dass die Konzepte von Bold es grundsätzlich erlauben würden, ein Modell in mehrere

---

[72] Vgl. Herrmann, Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper 2015, S. 5
[73] Vgl. Herrmann, Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper 2015, S. 14
[74] Vgl. Herrmann, Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper 2015

Teilmodelle aufzuteilen. In der jetzigen offiziellen Version von Bold for Delphi ist dies aber nicht möglich.

4.5.2 MDriven

Der Modularisierungsthematik hat man sich beim Nachfolger MDriven angenommen. In einem der Beispielprojekte wird aufgezeigt, wie man Klassen aus zwei unterschiedlichen Modellen in ein drittes Modell einbindet, um so mehrere Modelle zu einem neuen Modell zusammenzuführen. Die Klassen können nicht um weitere Attribute oder Operationen erweitert werden; Beziehungen zu anderen Klassen aus anderen Modellen sind aber möglich – sowohl Vererbungsbeziehungen wie auch Assoziationen. Die Assoziationen werden dabei über eine spezielle ebenfalls generierte Verbindungklasse gelöst.

Zusätzlich gibt es die Möglichkeit, mithilfe der PersistenceMapperMultiDb-Komponente, die Persistenzschicht auf mehrere Datenbanken aufzuteilen. Die Aufteilung erfolgt klassenweise. Es kann dann für jede Klasse genau eine Datenbank angegeben werden. Assoziationen zwischen Klassen in verschiedenen Datenbanken sind möglich.

Durch eine Kombination dieser beiden Verfahren kann eine gute Modularisierbarkeit gewährleistet werden. Lediglich die Erweiterung von Klassen aus anderen Modellen ist nicht möglich, was sich aber über Aggregationen bzw. Kompositionen pragmatisch lösen lässt.

4.5.3 Texo

Die Fähigkeit, mehrere Modelle zusammenzuführen, besteht grundsätzlich mit dem EMF-Framework. Texo selbst nutzt diese Technik bereits, um das ECore-Modell mit Informationen aus dem „Annotations Model“ anzureichern (s. Kapitel 4.2.3 und 4.3.3). Auch zwei Teilmodelle zu einem Modell zusammenzuführen, sollte damit kein Problem sein. Allerdings stößt das Konzept der Modularisierbarkeit bei der Quellcode-Generierung an seine Grenzen. Der Quellcode kann nicht auf mehrere Module aufgeteilt, sondern nur als Ganzes generiert werden. Ebenso steht das Sprachkonzept der partiellen Klassen, wie es bei C# existiert, für Java nicht zur Verfügung.

Was mit der JPA bzw. EclipseLink wiederum möglich ist, ist die Aufteilung der Persistenzschicht auf mehrere Datenbanken. Dafür sind mehrere PersistenceUnits

(orm.xml) anzulegen – eine für jede Datenbank – und die Entitäten entsprechend aufzuteilen. Es ist dann allerdings erforderlich, dass Texo Kenntnis über die Module hat und entsprechend mehrere PersistenceUnits generiert. Dies wird jedoch so nicht von Texo unterstützt.

Zusammenfassend ist die Modularisierbarkeit mit Texo zunächst nicht möglich, sollte sich aber im Gegensatz zu Bold leichter umsetzen lassen, da viele der Voraussetzungen dank EMF und EclipseLink bereits erfüllt sind.

5 Bewertung des Vergleichsergebnisses und des modellgetriebenen Ansatzes bei O/R-Mappern im Allgemeinen

Der Vergleich hat einerseits gezeigt, dass alle untersuchten modellgetriebenen O/R-Mapping-Frameworks durchaus in der Praxis eingesetzt werden können und teilweise einen fortgeschrittenen Funktionsumfang bieten. Andererseits erfüllt keines der Frameworks alle Anforderungen. Es zeigten sich z. T. erhebliche Schwächen bei einzelnen Vergleichskriterien.

Eine genaue Diskussion und Bewertung der Vergleichsergebnisse liefert Kapitel 5.1. Im Anschluss daran wird in Kapitel 5.2 analysiert, ob sich die Erwartungen an den modellgetriebenen Ansatz in dem Vergleich erfüllt haben und ob sich der Einsatz von modellgetriebenen O/R-Mapping-Frameworks gegenüber codegetriebenen O/R-Mappern lohnt.

5.1 Diskussion der Vergleichsergebnisse

5.1.1 Gemeinsamkeiten

Beginnend mit den Gemeinsamkeiten konnten zunächst einmal alle Produkte installiert und verwendet werden. Das erstellte Beispielmodell konnte danach in allen modellgetriebenen O/R-Mappern umgesetzt werden. Dabei gab es nur marginale Unterschiede, die sich aber nicht auf den Funktionsumfang auswirkten. So waren z. B. die Datentypen z. T. an die Zielsprache gebunden. Ebenso konnte die Klasse OrderPosition in Bold und MDriven als Linkklasse einer Assoziation zwischen den Klassen Article und Order ausgestaltet werden. Die Quellcode-Generierung funktionierte in allen Fällen und erzeugte kompilierbaren Quellcode, sodass schließlich als Ergebnis eine Anwendung ausgeführt und getestet werden konnte.

Trotz einiger Unterscheide beim Funktionsumfang beherrschen alle O/R-Mapper die grundlegenden Funktionen der objektrelationalen Abbildung. Neben CRUD-Operationen gehören dazu auch Abfragen in einer speziellen objektorientierten Abfragesprache oder alternativ mit SQL. Ebenso ist die Navigation über Assoziationen inkl. Eager- und Lazy-Loading möglich. Eine grundlegende Transaktionskontrolle mit einfachen und geschachtelten Transaktionen sowie einen Cache bieten alle O/R-Mapper. Darüber hinaus sind immer ein Mehrbenutzerbetrieb und dazu passende

optimistische/pessimistische Locking-Mechanismen vorgesehen (bei Bold jedoch nicht komfortabel und nur über Erweiterungen).

Das Modell wird bei allen Produkten dazu eingesetzt, um den Quellcode und die Metadaten für das O/R-Mapping zu generieren. Bei der Generierung wird vom Entwickler manuell geschriebener Quellcode nicht wieder überschrieben. Will man die Mapping-Einstellungen nach individuellen Wünschen anpassen, ist dies grundsätzlich möglich: Entweder direkt im Modell (Bold und MDriven) oder über ein gesondertes Annotations Model (Texo). Parallel zum O/R-Mapping kann auch das Datenbankschema generiert werden. Alle Produkte unterstützen dabei immer mehrere, wenn auch unterschiedliche, relationale Datenbanksysteme. Schließlich bieten alle Frameworks spezielle Validierungsmöglichkeiten über das Modell an.

5.1.2 Unterschiede und Alleinstellungsmerkmale

Die getesteten Produkte unterschieden sich teils deutlich im Funktionsumfang, insbesondere dann, wenn es über das reine O/R-Mapping hinausgeht. Beim O/R-Mapping selbst gibt es nur kleine Unterschiede. Dazu gehören u. a. die verwendeten Abfragesprachen. Bei Texo bzw. dem eingesetzten JPA-Provider EclipseLink sind es JPQL bzw. die EclipseLink Expressions, beide stark an SQL angelehnt. Bold und MDriven nutzen einheitlich OCL sowohl für Abfragen wie auch für Constraints im Modell. Bei den Transaktionen unterstützt Texo (bzw. EclipseLink) als einziger O/R-Mapper parallele Transaktionen. Die Möglichkeit, mehrere Versionsstände in der Datenbank zu speichern, bieten wiederum nur Bold und MDriven. Obwohl EclipseLink eigentlich diese Funktion unterstützt, sieht Texo dies bei der Generierung hingegen nicht vor. Die Mandantenfähigkeit (Multitenancy) lässt sich bei MDriven und Texo pro Klasse einstellen, Bold kennt diese Funktionalität hingegen nicht.

Die Nutzung des Modells als Kern der Anwendung wird bei MDriven am deutlichsten. Hier kann nicht nur die Klassenstruktur und das O/R-Mapping aus dem Modell heraus generiert werden, sondern auch das Verhalten in Form von Zustandsdiagrammen und durch die Benutzung der EAL für Operationen. Zusätzlich kann der Aufbau und das Layout der Benutzeroberfläche im Modell gestaltet werden. Damit ist MDriven als einziges Produkt in der Lage, ein Modell direkt auszuführen, was für das Prototyping sehr nützlich ist und einen wichtigen Schritt in Richtung Executable UML bedeutet.

MDriven bietet hier also ein Alleinstellungsmerkmal – bei keinem anderen Produkt gelingt die Verzahnung zwischen den verschiedenen Modellelementen so gut.

Demgegenüber steht allerdings die fehlende Integration in den Entwicklungszyklus der MDA. MDriven bietet im Gegensatz zu Texo keine Möglichkeit XMI-Dateien oder Modelle in irgendeiner Form zu importieren. Texo stehen hier dank der Verwendung von EMF viele Möglichkeiten zur Verfügung und es werden die XMI Version 2 und die UML Versionen 1.4/2.x unterstützt. Bold bleibt hier ebenfalls außen vor, da lediglich die veraltete XMI Version 1 und UML 1.3 genutzt werden können. Ein weiteres Alleinstellungsmerkmal von Texo ist die Möglichkeit der automatisierten Generierung von Testdaten durch die Modellinformationen.

Die Funktion der Database Evolution wird von Texo nicht unterstützt, es können lediglich neue Tabellen und Spalten automatisch hinzugefügt werden. In der Klassenhierarchie verschobene Klassen oder Attribute erkennen nur Bold und MDriven. In vielen Situationen können hier die automatisch generierten DDL-Skripte verwendet werden, ohne dass ein Datenverlust droht.

Die Modularisierung von Systemen wird nur von MDriven unterstützt. Bei Texo und Bold wären dafür umfangreiche manuelle Anpassungen erforderlich. Hierbei wurde auch ein einziges Mal ein Unterschied zwischen den verwendeten Programmiersprachen deutlich, da nur C# partielle Klassen unterstützt und damit die Modularisierung vereinfacht.

Zuletzt gibt es noch Unterschiede bei der Lizensierung: Bold for Delphi ist mittlerweile kostenlos mit der Delphi IDE (kostenpflichtig) erhältlich, während der Nachfolger MDriven kostenpflichtig und Closed Source ist (Nutzung als SaaS ist ebenfalls möglich); Texo ist als einziges Produkt wirklich vollkommen Open Source und kostenlos unter der EPL nutzbar.

5.1.3 Gegenüberstellung

Nachdem in den beiden vorherigen Kapiteln die Gemeinsamkeiten und Unterschiede aufgezählt wurden, folgt nun eine Gegenüberstellung und Bewertung der verglichenen Produkte. Dabei soll aufgezeigt werden, für welches Einsatzszenario welches Framework am besten geeignet ist.

Die Kerndisziplinen der ORM beherrschen alle drei verglichenen modellgetriebenen O/R-Mapper fast gleichermaßen. Bold for Delphi merkt man jedoch an, dass die letzte Version vor über 10 Jahren veröffentlich wurde. Es ist noch nicht auf eine gute Skalierbarkeit und den Einsatz mit vielen Benutzern gleichzeitig bzw. auf die Mandantenfähigkeit ausgelegt. Von der Nutzung als Webdienst ist eher abzuraten. Für den Einsatz in klassischen zweischichtigen Client-Server-Architekturen ist Bold nach wie vor geeignet. Bold war und ist ein mächtiges Framework, insbesondere beim Funktionsumfang, der über das O/R-Mapping hinausgeht. Betrachtet man die reinen ORM-Funktionen liegt es heute allerdings weit hinter den Möglichkeiten des JPA-Standards zurück und ist schon fast als ein leichtgewichtiger O/R-Mapper zu betrachten. Als Framework bietet es aber einige Funktionen (darunter OCL-Ausdrücke und der Subscription-Mechanismus), die es sehr einfach machen, einen Einstieg in die MDSD zu finden und Anwendungen mit diesem Framework zu erstellen. Auch der erste Kontakt des Autors mit Bold vor der Anfertigung dieser Arbeit verlief wesentlich leichter als mit den beiden anderen Produkten, die für diesen Vergleich zum ersten Mal verwendet wurden.

MDriven hat die Konzepte von Bold konsequent weiterentwickelt und bietet mit der Definition von Zustandsautomaten, dem Design der Benutzeroberfläche und der Ausführung des Modells einige Alleinstellungsmerkmale. Dabei wird wie bei Bold immer die von der OMG standardisierte OCL verwendet, bzw. die daran angelehnte EAL. Texo bietet für Abfragen stattdessen die JPQL bzw. EclipseLink Expressions an. Diese werden zwar objektorientiert verwendet, die Nähe zu SQL macht aber die Herkunft deutlich. Die OCL ist konsequenter auf die objektorientierte Sichtweise ausgerichtet; Kenntnisse des relationalen Modells sind dann nicht mehr erforderlich. Erweiterungen für das EMF ermöglichen zwar die Verwendung von OCL in ECore-Modellen, allerdings bietet Texo hier keine Unterstützung.

Ein Modell nutzen alle drei Frameworks und sind damit dem MDSD zuzuordnen. Der MDA-Ansatz wird aber lediglich bei Texo wirklich verfolgt, auch wenn die anderen Produkte mit MDA werben. Nur mit Texo lassen sich mehrere Abstraktionsstufen eines Modells realisieren und nur bei diesem Framework wird ein Ausgangsmodell mit den für Texo relevanten Metadaten angereichert. Bei den anderen Frameworks wird stattdessen eine starke Bindung an das jeweilige Produkt erzeugt; ein Wechsel auf ein anderes Framework ist damit nicht mehr so einfach möglich. Besonders negativ

fiel hier die fehlende Importmöglichkeit auf: Das Beispielmodell, welches mit einer externen Software erstellt wurde, musste in Bold komplett neu erstellt werden und konnte anschließend zumindest nach MDriven exportiert werden. In Texo war der Import hingegen problemlos möglich. Texo bietet zudem mit der automatischen Generierung von Testdaten eine weitere nützliche Funktion für das Prototyping.

Fazit Bold for Delphi: Nach Meinung des Autors ist Bold for Delphi auch heute noch ein Framework, das sich lohnt auszuprobieren. Es bietet viele gute Konzepte und vor Allem einen sehr einfachen Einstieg in die modellgetriebene Entwicklung, insbesondere dann, wenn man mit der Programmiersprache Delphi arbeitet. Zur weiterführenden Nutzung wird es aber nicht mehr empfohlen.

Fazit MDriven: Nach Meinung des Autors bietet MDriven den weitaus größten und abgerundetsten Funktionsumfang der verglichenen Produkte und ein weites Spektrum an Einsatzmöglichkeiten, allerdings auf Kosten der Standardisierung. Akzeptiert man die Bindung an das Framework und den Verzicht auf wirkliche MDA und ist man zudem bereit, die Lizenzkosten zu zahlen, hat man mit MDriven eine gute Wahl im .Net-Umfeld getroffen.

Fazit Texo: Nach Meinung des Autors trifft man mit Texo die beste Wahl, wenn es darum geht, plattformunabhängige Webdienste in Java zu entwickeln und dabei zukunftssicher den MDA-Ansatz zu verfolgen. Mit der JPA stehen dabei auch die umfangreichsten ORM-Funktionalitäten und eine gute Skalierbarkeit zur Verfügung.

Zusammenfassend zeigt die folgende Tabelle 5 eine Bewertungsmatrix, mit den eben hervorgehobenen Kriterien Einstieg (Leicht, Mittel oder Schwer), Umfang (Gering, Mittel oder Groß), Einsatzgebiet (Client/Server, N-Tier oder Webdienst) und Umsetzung der MDA (ja oder nein):

Tabelle 5: Vereinfachte Bewertungsmatrix der verglichenen O/R-Mapper

Framework	Einstieg	Umfang	Einsatzgebiet	MDA
Bold	Leicht	Mittel	Client/Server	Nein
MDriven	Mittel	Groß	Client/Server, N-Tier & Webdienste	Nein
Texo	Mittel	Mittel	Webdienste	Ja

5.2 Vor- und Nachteile des modellgetriebenen Ansatzes im Bereich der O/R-Mapper

Dieses Kapitel ist allgemein auf den modellgetriebenen Ansatz bezogen und soll nicht die Alleinstellungsmerkmale oder Schwachstellen einzelner Produkte einbeziehen, welche bereits im vorherigen Kapitel deutlich gemacht wurden. Der Vergleich konnte im Wesentlichen die in Kapitel 2.3.3 vermuteten Vor- und Nachteile von modellgetriebenen O/R-Mapper bestätigen.

Allem voran war die Produktivitätssteigerung deutlich. Nach einer gewissen Einarbeitungszeit musste lediglich das Modell in den jeweiligen Frameworks importiert bzw. erstellt werden. Daraufhin erhielt man bereits ausführbaren Quellcode, wenn auch ohne Logik. Manuelle Implementierungen waren minimal und dienten lediglich der Initialisierung der Anwendung. Es war als Test bei allen Frameworks möglich, Objekte zu erzeugen und zu verknüpfen sowie Attribute zu ändern. Für die Persistenz mussten neben der Datenbankanbindung keinerlei Einstellungen vorgenommen werden und es waren keine Kenntnisse über das relationale Modell, SQL oder die objektrelationale Abbildung bzw. den verwendeten O/R-Mapper selbst erforderlich. Die Speicherung erfolgte konsistent und ohne Fehler im Hintergrund, sodass hier ebenso die angekündigte Qualität der generierten Artefakte bestätigt werden konnte.

Allerdings sind die Kenntnisse über die generierten Artefakte nach wie vor wichtig, da nur mit dem Wissen über diese Konzepte das Modell bzw. das O/R-Mapping später optimiert gestaltet werden kann. Die MDSD darf also nicht dazu verleiten, sich nicht mit den zugrundeliegenden Konzepten zu beschäftigen, auch wenn Generatoren dem Entwickler einen Teil der Arbeit abnehmen. Eine Vernachlässigung wirkt sich daher negativ auf die Performance und auf die Qualität des Systems aus. Diese Gefahr besteht z. T. aber bereits bei codegetriebenen O/R-Mappern, da auch hier SQL-Abfragen im Hintergrund stattfinden und diese standardmäßig nicht optimiert sind. Mit jeder weiteren Abstraktionsstufe werden also Implementierungsdetails der vorhergehenden Stufe verborgen.

Auch die geringere Flexibilität konnte im Vergleich bestätigt werden. Ein Beispiel bildet hier die fehlende Unterstützung Texos für die Versionierung, obwohl der zugrundeliegende O/R-Mapper diese Möglichkeit bietet. Ebenso lässt sich das Problem der Modularisierung von Modellen bei den verglichenen Produkten noch nicht vollständig umsetzen, auch wenn MDriven hier fast alle Anforderungen erfüllt.

6 Zukünftige Bedeutung modellgetriebener O/R-Mapper

In dieser Arbeit wurde der Begriff des modellgetriebenen O/R-Mappers bzw. des modellgetriebenen O/R-Mapping-Frameworks definiert und erläutert. Dabei handelt es sich zusammenfassend um ein Softwareframework, das sich – ganz im Sinne der MDSD – mit einem Modell im Mittelpunkt in erster Linie um die Aufgabe der objektrelationalen Abbildung kümmert, darüber hinaus aber das Modell zusätzlich für weitere Zwecke nutzt.

Die Vorteile des modellgetriebenen Ansatzes im Vergleich zur alleinigen Nutzung eines codegetriebenen O/R-Mappers sind, wie für die MDSD üblich, gesteigerte Produktivität und Qualität der erzeugten Artefakte. Zusätzlich lässt sich das Modell, welches sowieso während der Entwurfsphase zum Austausch mit den Stakeholdern erstellt werden sollte, für die Generierung weiterer Artefakte nutzen. Voraussetzung ist dabei ein formalisiertes Modell. Mit der Persistenz muss sich der Softwarearchitekt oder Entwickler theoretisch nicht mehr auseinandersetzen. Kenntnisse über die generierten Artefakte und der dahinterliegenden Konzepte sind aber nach wie vor erforderlich, wenn es um die Optimierung geht. Dies gilt vor Allem für die Einstellungen des O/R-Mappings, die man bei der codegetriebenen Variante immer manuell festlegen muss.

Der größte Nachteil ist die geringere Flexibilität, die man z. B. mit der Bindung an ein Framework eingeht. Vorteilhaft ist hier die Verwendung des MDA-Standards, damit man zukunftssicher und zunächst plattformunabhängig das Modell entwerfen kann. Oft ist auch der Funktionsumfang gegenüber den codegetriebenen O/R-Mappern begrenzt, da es keine Entsprechung für alle Funktionen des O/R-Mappings im Modell gibt. Hochoptimierte Systeme lassen sich mit dem modellgetriebenen Ansatz wahrscheinlich nicht umsetzen. Das ist aber auch nicht das Ziel, sondern eher der schnelle und einfache Weg zur fertigen Anwendung. Idealerweise verwendet man den Ansatz deshalb bei neuen Systemen. Die Umstellung eines bestehenden Systems auf die Verwendung eines modellgetriebenen Frameworks kann möglicherweise aufwendig und schwierig sein.

Der Vergleich mit drei in der Praxis eingesetzten modellgetriebenen O/R-Mapping-Frameworks hat diese Thesen bestätigt. Die Produkte am Markt bieten teilweise einen großen Funktionsumfang, der weit über das O/R-Mapping hinausgeht. Die

Einarbeitung in die grundlegenden Techniken ging meist schnell und die Erstellung einer Testanwendung war danach problemlos möglich. Die größte Schwäche offenbarte sich da, wo die Standardisierung vernachlässigt wurde. Lediglich ein Framework bot Unterstützung für den MDA-Standard. Auch die fehlende Flexibilität konnte beim Vergleich festgestellt werden. Hält man sich an die Vorgaben und Konzepte des Frameworks erzielt man schnell gute Ergebnisse. Individualisierungen oder Abweichungen von diesen Konzepten sind hingegen nur schwer möglich.

Zudem ist aufgefallen, dass es – im Vergleich zu den codegetriebenen Varianten – am Markt nur wenige O/R-Mapper gibt, die ein Modell als Basis verwenden. Viele davon sind zudem veraltet und werden nicht mehr weiterentwickelt. Selbst der MDA-Standard scheint trotz der Vorteile nur langsam vorangetrieben zu werden. Vielleicht ändert sich dies, wenn mit der UML auch Datenbankmodelle entworfen werden können. Bisher sieht die UML keine Sprachelemente für das relationale Datenbankmodell oder so etwas wie ein Persistenzdiagramm vor. Die Einführung solche Elemente, insbesondere dann, wenn sie mit den anderen Modellelementen wie z. B. Klassen verknüpft werden können, würde den Einsatz des MDSD im Bereich der Persistenz weiter vorantreiben. Die objektrelationalen Erweiterungen am Modell, die jedes der Frameworks in Form von tagged values, eigenen UML-Profilen oder auf andere Weise vornehmen muss, wären dann standardisiert und nicht mehr abhängig vom Framework und dem zugrundeliegenden O/R-Mapper. Möglicherweise wäre damit das ER-Modell überflüssig oder würde in der UML aufgehen. Eine Integration des ER-Modells in die UML eröffnet damit neue Forschungsmöglichkeiten. Sobald es in dieser Richtung Standards gibt, wird auch zukünftig die Bedeutung von modellgetriebenen O/R-Mappern wachsen.

Literaturverzeichnis

AndroMDA.org. *AndroMDA Model Driven Architecture Framework – AndroMDA - Homepage.* 2014. <http://andromda.sourceforge.net/> (Zugriff am 12. 05 2016).

AthenaSource. *Athena Framework for Java Overview - Athena Framework.* o. J. <http://www.athenasource.org/java/index.php> (Zugriff am 16. 03 2016).

BoldSoft MDE AB. *BoldSoft | Products | White Papers | Enlighten me.* 15. 08 2002. <http://web.archive.org/web/20020815020327/http://www.boldsoft.com/products/whitepapers/enlighten_me.html> (Zugriff am 15. 05 2016).

CapableObjects AB. *MDriven – the book | CapableObjects.* o. J. <http://www.new.capableobjects.com/products/mdriven-the-book/> (Zugriff am 15. 05 2016).

—. *MDriven Framework – Model driven framework formerly known as ECO | CapableObjects.* o. J. <http://www.new.capableobjects.com/products/eco-model-driven-framework/> (Zugriff am 16. 03 2016).

—. *MDriven Turnkey | CapableObjects.* o. J. <http://www.new.capableobjects.com/turnkey/> (Zugriff am 29. 05 2016).

Devart. *ORM for Delphi with LINQ Support.* 2016. <https://www.devart.com/entitydac/> (Zugriff am 12. 05 2016).

Eclipse Foundation. *Database Support | EclipseLink 2.6.x Understanding EclipseLink.* 27. 04 2015. <http://www.eclipse.org/eclipselink/documentation/2.6/concepts/app_tl_ext001.htm> (Zugriff am 25. 05 2016).

—. *Eclipse Modeling Project.* o. J. <http://www.eclipse.org/modeling/emf/> (Zugriff am 12. 05 2016).

—. *Introduction to EclipseLink Transactions (ELUG) - Eclipsepedia.* 23. 07 2012. <https://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_(ELUG)#Nested_and_Parallel_Units_of_Work> (Zugriff am 05. 06 2016).

—. *Texo - Eclipsepedia.* 08. 12 2015. <http://wiki.eclipse.org/Texo> (Zugriff am 12. 05 2016).

—. *Texo/Download and Install - Eclipsepedia.* 20. 11 2013. <http://wiki.eclipse.org/Texo/Download_and_Install> (Zugriff am 15. 05 2016).

—. *Texo/Texo and EMF - Eclipsepedia.* 16. 05 2010. <http://wiki.eclipse.org/Texo/Texo_and_EMF> (Zugriff am 20. 05 2016).

Edlich, Stefan, Achim Friedland, Jens Hampe, Benjamin Brauer, und Markus Brückner. *NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken.* 2. akt. & erw. Auflage. München: Hanser Verlag, 2011.

Faeskorn-Woyke, Heide, Birgit Bertelsmeier, Petra Riemer, und Elena Bauer. *Datenbanksysteme - Theorie und Praxis mit SQL2003, Oracle und MySQL.* München: Pearson, 2007.

Flores Beltran, Juan Carlos, et al. *Modellgetriebene Softwareentwicklung - MDA und MDSD in der Praxis.* Herausgeber: Georg Pietrek und Jens Trompeter. Frankfurt (Main): Entwickler.press, 2007.

Fowler, Martin. *OrmHate.* 08. 05 2012. <http://martinfowler.com/bliki/OrmHate.html> (Zugriff am 04. 06 2016).

Geisler, Frank. *Datenbanken - Grundlagen und Design.* 5. aktual. u. erw. Auflage. Heidelberg: mitp Verlag, 2014.

Generative Software Engineering Zwickau. *GeneSEZ: Welcome to GeneSEZ.* o. J. <http://www.genesez.org/> (Zugriff am 12. 05 2016).

Grose, Timothy J., Gary C. Doney, und Stephen A. Brodsky. *Mastering XMI - Java Programming with XMI, XML, and UML.* New York: Wiley Computer Publishing, 2002.

Herrmann, Christian. „Konzept zur Modularisierung von Geschäftsmodellen für modellgetriebene O/R-Mapper.“ 26. 10 2015.

Hibernate. *Hibernate ORM - Hibernate ORM.* o. J. <http://hibernate.org/orm/> (Zugriff am 28. 05 2016).

Holder, Stefan, Jim Buchan, und Stephen G. MacDonell. „Towards a Metrics Suite for Object-Relational Mappings.“ *Model-Based Software and Data Integration.* Berlin: Springer, 2008. S. 43-54.

Ireland, Christopher, und David Bowers. „Exposing the Myth: Object-Relational Impedance Mismatch is a Wicked Problem.“ *The Seventh International Conference on Advances in Databases, Knowledge and Data Applications,* 2015: S. 21-26.

Koegel, Maximlian, und Jonas Helming. *EMF Tutorial « EclipseSource Blog.* 14. 03 2016. <http://eclipsesource.com/blogs/tutorials/emf-tutorial/> (Zugriff am 20. 05 2016).

Kornmeier, Martin. *Wissenschaftlich schreiben leicht gemacht für Bachelor, Master und Dissertation.* 6., aktualisierte Auflage. Bern: Haupt Verlag, 2013.

Laudon, Kenneth C., Jane P. Laudon, und Detlef Schoder. *Wirtschaftsinformatik - Eine Einführung.* 2., aktualisierte Auflage. München: Pearson Verlag, 2010.

Liddle, Stephen W. *Model-Driven Software Development.* Brigham, 2010.

Ludewig, Jochen, und Horst Lichter. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken.* 2. Auflage. Heidelberg: dpunkt-Verlag, 2010.

Mellor, Stephen J., und Marc J. Balcer. *Executable UML: a foundation for model-driven architecture.* Boston: Addison-Wesley, 2002.

Microsoft Corporation. *Design Time Code Generation and Runtime Model-Driven Generation.* 04 2010. <https://msdn.microsoft.com/en-us/library/ff621668.aspx> (Zugriff am 14. 08 2015).

—. *Entity Framework.* 2015. <https://msdn.microsoft.com/de-de/data/ef.aspx> (Zugriff am 28. 05 2016).

Müller, Bernd, und Harald Wehr. *Java Persistence API 2 - Hibernate, EclipseLink, OpenJPA und Erweiterungen.* München: Carl Hanser Verlag, 2012.

Neward, Ted. *The Vietnam of Computer Science.* 26. 06 2006. <http://blogs.tedneward.com/post/the-vietnam-of-computer-science/> (Zugriff am 04. 06 2016).

nHydrate. *GitHub - nHydrate.* 2016. <https://github.com/nHydrate/nHydrate> (Zugriff am 16. 03 2016).

—. *nHydrate Code Generation Platform - Documentation.* 01. 09 2010. <https://nhydrate.codeplex.com/wikipage?title=nhibernate> (Zugriff am 28. 05 2016).

Object Management Group. „OMG Document -- ormsc/14-06-01 (MDA Guide revision 2.0).“ Vers. 2. 01. 06 2014. <http://www.omg.org/cgi-bin/doc?ormsc/14-06-01> (Zugriff am 19. 2 2016).

Oracle Corporation. *Java Persistence API.* o. J. <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html> (Zugriff am 28. 05 2016).

Petrasch, Roland, und Oliver Meimberg. *Model Driven Architecture. Eine praxisorientierte Einführung in die MDA.* Heidelberg: dpunkt.verlag, 2006.

Pilone, Dan. *UML 2.0 in a Nutshell.* 2. Auflage. Sebastopol CA: O'Reilly Media, 2005.

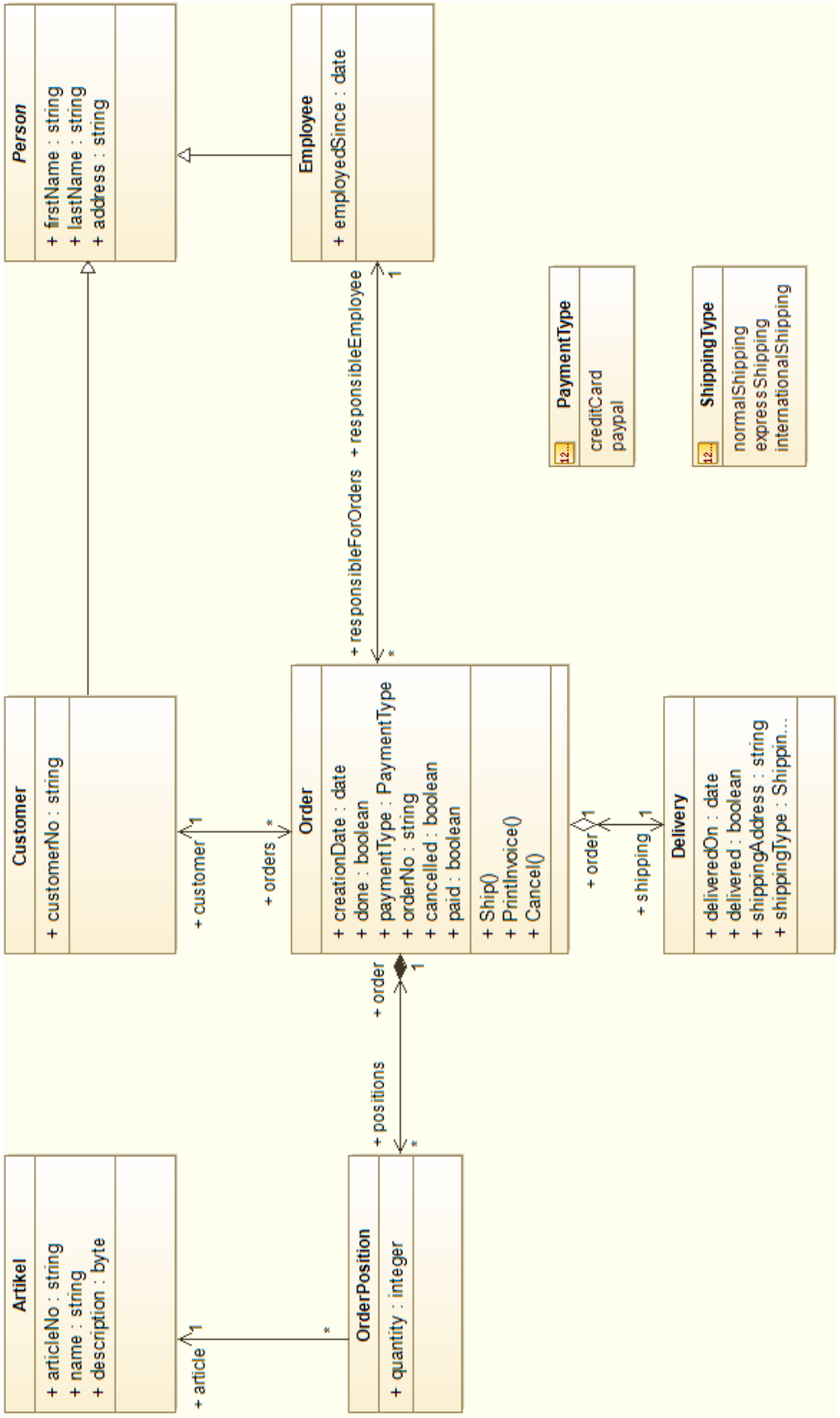
Rehn, Christian. *FoAM – A Framework for Application-Oriented O/R Mapping.* 30. 04 2007.

Sommerville, Ian. *Software Engineering.* Bd. 8. aktualisierte Aufl. München: Pearson Studium, 2007.

Warmer, Jos B., und Anneke G. Kleppe. *The Object Constraint Language - Precise Modeling with UML.* 3. Druck. Boston: Addison-Wesley, 2000.

Anhang

Vollständiges Beispielmodell



```
<?xml version="1.0" encoding="UTF-8"?><uml:Model
xmlns:uml="http://www.omg.org/spec/UML/20110701"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmi:version="2.1"
xmi:id="_TNWfICTzEeaNoaZJTkje1w" name="testmodel">
  <eAnnotations xmi:id="_TNWfISTzEeaNoaZJTkje1w" source="Objing">
    <contents xmi:type="uml:Property"
xmi:id="_TNXGMCTzEeaNoaZJTkje1w" name="exporterVersion">
      <defaultValue xmi:type="uml:LiteralString"
xmi:id="_TNXGMSTzEeaNoaZJTkje1w" value="3.0.0"/>
    </contents>
  </eAnnotations>
  <packagedElement xmi:type="uml:Class"
xmi:id="_TNXGMiTzEeaNoaZJTkje1w" name="Order">
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGMyTzEeaNoaZJTkje1w" name="positions"
visibility="public" type="_TNXGSiTzEeaNoaZJTkje1w"
aggregation="composite" association="_TNXGQiTzEeaNoaZJTkje1w">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_TNXGNCTzEeaNoaZJTkje1w" value="*"/>
      <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_TNXGNSTzEeaNoaZJTkje1w"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGNiTzEeaNoaZJTkje1w" name="responsibleEmployee"
visibility="public" type="_TNXGXCTzEeaNoaZJTkje1w"
association="_TNXGQyTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGNyTzEeaNoaZJTkje1w" name="shipping" visibility="public"
type="_TNXGYiTzEeaNoaZJTkje1w" aggregation="shared"
association="_TNXGRCTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGOCTzEeaNoaZJTkje1w" name="customer" visibility="public"
type="_TNXGUiTzEeaNoaZJTkje1w"
association="_TNXGRSTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGOSTzEeaNoaZJTkje1w" name="creationDate"
visibility="public" type="_TNXGbyTzEeaNoaZJTkje1w"
isUnique="false"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGOiTzEeaNoaZJTkje1w" name="done" visibility="public"
isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#Boolea
n"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGOyTzEeaNoaZJTkje1w" name="paymentType"
visibility="public" type="_TNXGaCTzEeaNoaZJTkje1w"
isUnique="false"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGPCTzEeaNoaZJTkje1w" name="orderNo" visibility="public"
isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGPSTzEeaNoaZJTkje1w" name="cancelled"
visibility="public" isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#Boolea
```

```
n"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGPiTzEeaNoaZJTkje1w" name="paid" visibility="public"
isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#Boolea
n"/>
    </ownedAttribute>
    <ownedOperation xmi:type="uml:Operation"
xmi:id="_TNXGPyTzEeaNoaZJTkje1w" name="Ship" visibility="public"/>
    <ownedOperation xmi:type="uml:Operation"
xmi:id="_TNXGQCTzEeaNoaZJTkje1w" name="PrintInvoice"
visibility="public"/>
    <ownedOperation xmi:type="uml:Operation"
xmi:id="_TNXGQSTzEeaNoaZJTkje1w" name="Cancel" visibility="public"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Association"
xmi:id="_TNXGQiTzEeaNoaZJTkje1w" memberEnd="_TNXGMyTzEeaNoaZJTkje1w
_TNXGSyTzEeaNoaZJTkje1w"/>
  <packagedElement xmi:type="uml:Association"
xmi:id="_TNXGQyTzEeaNoaZJTkje1w" memberEnd="_TNXGNiTzEeaNoaZJTkje1w
_TNXGXiTzEeaNoaZJTkje1w"/>
  <packagedElement xmi:type="uml:Association"
xmi:id="_TNXGRCTzEeaNoaZJTkje1w" memberEnd="_TNXGNyTzEeaNoaZJTkje1w
_TNXGYyTzEeaNoaZJTkje1w"/>
  <packagedElement xmi:type="uml:Association"
xmi:id="_TNXGRSTzEeaNoaZJTkje1w" memberEnd="_TNXGOCTzEeaNoaZJTkje1w
_TNXGVCTzEeaNoaZJTkje1w"/>
  <packagedElement xmi:type="uml:Class"
xmi:id="_TNXGRiTzEeaNoaZJTkje1w" name="Artikel">
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGRyTzEeaNoaZJTkje1w" name="articleNo"
visibility="public" isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGSCTzEeaNoaZJTkje1w" name="name" visibility="public"
isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGSSTzEeaNoaZJTkje1w" name="description"
visibility="public" type="_TNXGcCTzEeaNoaZJTkje1w"
isUnique="false"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Class"
xmi:id="_TNXGSiTzEeaNoaZJTkje1w" name="OrderPosition">
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGSyTzEeaNoaZJTkje1w" name="order" visibility="public"
type="_TNXGMiTzEeaNoaZJTkje1w"
association="_TNXGQiTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGTCTzEeaNoaZJTkje1w" name="article" visibility="public"
type="_TNXGRiTzEeaNoaZJTkje1w"
association="_TNXGTiTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
```

```
xmi:id="_TNXGTSTzEeaNoaZJTkje1w" name="quantity" visibility="public"
isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#Intege
r"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Association"
xmi:id="_TNXGTiTzEeaNoaZJTkje1w" memberEnd="_TNXGTCTzEeaNoaZJTkje1w
_TNXGTyTzEeaNoaZJTkje1w">
    <ownedEnd xmi:type="uml:Property"
xmi:id="_TNXGTyTzEeaNoaZJTkje1w" visibility="public"
type="_TNXGSiTzEeaNoaZJTkje1w"
association="_TNXGTiTzEeaNoaZJTkje1w">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_TNXGUCTzEeaNoaZJTkje1w" value="*"/>
      <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_TNXGUSTzEeaNoaZJTkje1w"/>
    </ownedEnd>
  </packagedElement>
  <packagedElement xmi:type="uml:Class"
xmi:id="_TNXGUiTzEeaNoaZJTkje1w" name="Customer">
    <generalization xmi:type="uml:Generalization"
xmi:id="_TNXGUyTzEeaNoaZJTkje1w" general="_TNXGWCTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGVCTzEeaNoaZJTkje1w" name="orders" visibility="public"
type="_TNXGMiTzEeaNoaZJTkje1w"
association="_TNXGRSTzEeaNoaZJTkje1w">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_TNXGVSTzEeaNoaZJTkje1w" value="*"/>
      <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_TNXGViTzEeaNoaZJTkje1w"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGVyTzEeaNoaZJTkje1w" name="customerNo"
visibility="public" isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Class"
xmi:id="_TNXGWCTzEeaNoaZJTkje1w" name="Person" isAbstract="true">
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGWSTzEeaNoaZJTkje1w" name="firstName"
visibility="public" isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGWiTzEeaNoaZJTkje1w" name="lastName" visibility="public"
isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGWyTzEeaNoaZJTkje1w" name="address" visibility="public"
isUnique="false">
      <type xmi:type="uml:PrimitiveType"
```

```
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Class"
xmi:id="_TNXGXCTzEeaNoaZJTkje1w" name="Employee">
    <generalization xmi:type="uml:Generalization"
xmi:id="_TNXGXSTzEeaNoaZJTkje1w" general="_TNXGWCTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGXiTzEeaNoaZJTkje1w" name="responsibleForOrders"
visibility="public" type="_TNXGMiTzEeaNoaZJTkje1w"
association="_TNXGQyTzEeaNoaZJTkje1w">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural"
xmi:id="_TNXGXyTzEeaNoaZJTkje1w" value="*"/>
      <lowerValue xmi:type="uml:LiteralInteger"
xmi:id="_TNXGYCTzEeaNoaZJTkje1w"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGYSTzEeaNoaZJTkje1w" name="employedSince"
visibility="public" type="_TNXGbyTzEeaNoaZJTkje1w"
isUnique="false"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Class"
xmi:id="_TNXGYiTzEeaNoaZJTkje1w" name="Delivery">
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGYyTzEeaNoaZJTkje1w" name="order" visibility="public"
type="_TNXGMiTzEeaNoaZJTkje1w"
association="_TNXGRCTzEeaNoaZJTkje1w"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGZCTzEeaNoaZJTkje1w" name="deliveredOn"
visibility="public" type="_TNXGbyTzEeaNoaZJTkje1w"
isUnique="false"/>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGZSTzEeaNoaZJTkje1w" name="delivered"
visibility="public" isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#Boolea
n"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGZiTzEeaNoaZJTkje1w" name="shippingAddress"
visibility="public" isUnique="false">
      <type xmi:type="uml:PrimitiveType"
href="http://www.omg.org/spec/UML/20110701/PrimitiveTypes.xmi#String
"/>
    </ownedAttribute>
    <ownedAttribute xmi:type="uml:Property"
xmi:id="_TNXGZyTzEeaNoaZJTkje1w" name="shippingType"
visibility="public" type="_TNXGayTzEeaNoaZJTkje1w"
isUnique="false"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Enumeration"
xmi:id="_TNXGaCTzEeaNoaZJTkje1w" name="PaymentType">
    <ownedLiteral xmi:type="uml:EnumerationLiteral"
xmi:id="_TNXGaSTzEeaNoaZJTkje1w" name="creditCard"/>
    <ownedLiteral xmi:type="uml:EnumerationLiteral"
xmi:id="_TNXGaiTzEeaNoaZJTkje1w" name="paypal"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Enumeration"
xmi:id="_TNXGayTzEeaNoaZJTkje1w" name="ShippingType">
    <ownedLiteral xmi:type="uml:EnumerationLiteral"
```

```
xmi:id="_TNXGbCTzEeaNoaZJTkje1w" name="normalShipping"/>
    <ownedLiteral xmi:type="uml:EnumerationLiteral"
xmi:id="_TNXGbSTzEeaNoaZJTkje1w" name="expressShipping"/>
    <ownedLiteral xmi:type="uml:EnumerationLiteral"
xmi:id="_TNXGbiTzEeaNoaZJTkje1w" name="internationalShipping"/>
  </packagedElement>
  <packagedElement xmi:type="uml:PrimitiveType"
xmi:id="_TNXGbyTzEeaNoaZJTkje1w" name="date"/>
  <packagedElement xmi:type="uml:PrimitiveType"
xmi:id="_TNXGcCTzEeaNoaZJTkje1w" name="byte"/>
</uml:Model>
```

Eidesstattliche Erklärung

Ich versichere an Eides Statt, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.