Institut für Softwaretechnologie
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie Nr. 147

Leichtgewichtige Java-OR-Mapping-Werkzeuge

Christian Buchgraber, Philipp Gildein, Philipp Pirrung

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Jochen Ludewig
Betreuer:	Daniel Kulesz, M.Sc.
begonnen am:	1. Juni 2011
beendet am:	24. November 2011
CR-Klassifikation:	D.2.3

Zusammenfassung

OR-Mapper (objektrelationale Mapper) sind heutzutage eine wichtige Abstraktionsschicht, um objektorientierte Konzepte in die Welt der relationalen Datenbanken zu bringen. Diese Werkzeuge bieten aber weit mehr als nur die Vermittlung zwischen den beiden Konzepten. Sie versuchen auch viele weitere Hilfestellungen zu leisten, von denen der Entwickler oftmals nur Bruchteile benötigt. So werden große Werkzeuge eingesetzt, um ein für den Entwickler kleines Problem zu lösen. Dieses Problem greift die Klasse der leichtgewichtigen objektrelationalen Mapper an. Durch ein kleineres Repertoire an Fähigkeiten und weniger externer Abhängigkeiten versuchen sie, einen schlankeren Lösungsansatz zu bieten.

In dieser Fachstudie soll ein Blick auf eben diese Klasse von Werkzeugen geworfen werden, um für ein konkretes, leichtgewichtiges Nutzungsszenario das am besten passende Werkzeug zu finden. Dazu wird zuerst ein genereller Überblick über die am Markt angebotenen OR-Mapper erstellt. Aus diesen werden dann leichtgewichtige Kandidaten ausgesucht und gemäß dem Nutzungsszenario evaluiert, um die für das spezielle Problem beste Lösung zu finden.

Abstract

Leightweight Java OR Mapping Tools

In today's world, OR mappers (object-relational mappers) form an important layer of abstraction used to integrate object-oriented concepts into relational databases. These tools offer a much broader range of features than merely the connection between the two concepts as they try to assist the user with many more problems. Developers generally only need a small part of the supplied features, hence using large-scale tools for small-scale problems. This issue is dealt with by the class of lightweight object-relational mappers. By offering a less extensive selection of features as well as fewer external dependencies, they aim for a slimmer approach towards the issue.

In the context of this study, this class of tools will be focused in an attempt to find the best-fitting tool for a specific lightweight user scenario. For this purpose an overview of OR mappers currently on the market will be created. The lightweight candidates will then be isolated and evaluated according to the user scenario. This finally results in a fitting tool for the specific problem.

Inhaltsverzeichnis

1 Einleitung

In der heutigen Softwareentwicklung sind objektorientierte Programmiersprachen kaum noch wegzudenken. Auf Sprachkonzepte wie Information Hiding oder Vererbung möchten nur die wenigsten bei der Entwicklung moderner Anwendungen verzichten. Ebenso sind relationale Datenbanken seit Jahrzehnten die erste Wahl für die persistente Datenspeicherung. Das Problem, dass sich objektorientierte Datenstrukturen nicht direkt in allen Aspekten mit relationalen Datenbanken abbilden lassen, wurde in den 1990er Jahren als *Object-relational Impedance Mismatch* [Ors06, Sch10] bekannt. Hierzu existieren verschiedene Lösungsansätze, um den konzeptionellen Widerspruch aufzulösen oder zumindest zu mildern. Ein Ansatz sind die sogenannten objektrelationalen Mapper, um die es in dieser Fachstudie geht.

1.1 Begriffe

OR-Mapper

Ein objektrelationaler Mapper („OR-Mapper“) ist nach [Rus08] eine Abstraktionsschicht, die das Objektmodell einer Anwendung mit einer relationalen Datenbank verbindet. Ein OR-Mapper lässt Datenbankzugriffe auf eine objektorientierte Weise zu und kümmert sich um die Zuordnung zwischen den Objekten der Anwendung und den Tabellen der Datenbank. Die OR-Zuordnung ist für den Entwickler unsichtbar.

Leichtgewichtigkeit

Im Kontext dieser Arbeit ist mit Leichtgewichtigkeit gemeint, dass wenige Abhängigkeiten zu externen Bibliotheken vorhanden sein dürfen.

1.2 Entstehungsgeschichte

2010 entwickelten studentische Hilfskräfte im Auftrag der Abteilung Software Engineering am Institut für Softwaretechnologie (ISTE) die Bibliothek *AdoHive*, die auf den ersten Blick stark an einen OR-Mapper erinnert. *AdoHive* übernimmt zwar einige Aufgaben eines OR-Mappers, schreibt aber das Datenmodell fest vor, weil die Bibliothek speziell für das Softwarepraktikum (SoPra) im selben Jahr entstanden ist. Die Teilnehmer des SoPras hatten die Aufgabe, ein Verwaltungssystem für die Hilfskräfte am Institutsverbund Informatik zu entwickeln und sollten zur Datenverwaltung auf die spezifische API von *AdoHive* zurückgreifen, sodass alle Datenbankzugriffe durch die Bibliothek gekapselt werden. Dadurch mussten sich die Studenten nicht in die Konzepte von relationalen Datenbanken einarbeiten, sondern konnten sich von Beginn an auf die Anwendungslogik und Benutzeroberfläche ihres Verwaltungssystems konzentrieren.

Als Teilnehmer an diesem Softwarepraktikum entwickelten wir die Software *aidGer* [BGP11], die bei der Abnahme als eines der Siegerprojekte gekürt wurde. Nach einer Evaluationsphase und der Migration der Daten aus dem vorherigen Verwaltungssystem ist die Software seitdem erfolgreich beim Kunden im Einsatz. Neue Anforderungen ver-

langen es nun, den Unterbau von *aidGer* anzupassen. Da sich *AdoHive* bei der Weiterentwicklung an *aidGer* als äußerst unflexibel offenbarte und die Wartung der Anwendung erschwerte, haben wir gemeinsam mit dem Kunden die Entscheidung getroffen, *AdoHive* durch einen frei verfügbaren OR-Mapper in der nächsten, stabilen Version zu ersetzen.

Zur selben Zeit suchte die Cinovo AG mit Sitz in Stuttgart ebenfalls einen OR-Mapper für ein neues Software-Projekt. Der gesuchte OR-Mapper soll dabei auf einem Gerät zum Einsatz kommen, das wenige Ressourcen zur Verfügung stellt. Durch diese Anforderung unterscheiden sich die Szenarien der beiden Projekte und können nicht von demselben Blickwinkel aus begutachtet werden. Daher konzentriert sich diese Fachstudie auf das Nutzungsszenario der Cinovo AG. Allerdings hoffen wir, dass uns die Erkenntnisse der Arbeit für eine Entscheidungsfindung bei der Wahl eines geeigneten OR-Mappers für das Verwaltungssystem *aidGer* weiterhelfen.

1.3 Wichtige Anforderungen

Die leistungsschwache Hardware erfordert einen speziellen OR-Mapper. Der Industriepartner stellt die folgenden Anforderungen an das gesuchte Werkzeug:

- ▷ Der gesuchte OR-Mapper muss in der Programmiersprache Java geschrieben sein.
- ▷ Der gesuchte OR-Mapper muss leichtgewichtig sein (s. 1.1).
- ▷ Der gesuchte OR-Mapper muss mit der Datenbank umgehen können, die bereits erfolgreich in anderen Software-Projekten des Unternehmens im Einsatz ist. Konkret handelt es sich um das DBMS PostgreSQL [PT11].
- ▷ Der gesuchte OR-Mapper muss aktiv weiterentwickelt werden und stabil laufen.
- ▷ Die Lizenz des gesuchten OR-Mappers darf den kommerziellen Einsatz nicht untersagen. Eine Open-Source-Lizenz wird vom Industriepartner bevorzugt.

1.4 Aufbau des Dokumentes

Kapitel 2 geht auf den generellen Ablauf der Fachstudie ein. Es werden die einzelnen Phasen der Studie beschrieben und diese auf einer Zeitachse eingeordnet. Einen Überblick über die momentan am Markt verfügbaren OR-Mapper liefert Kapitel 3. Das darauffolgende Kapitel behandelt das Nutzungsszenario der Cinovo AG und stellt das geforderte Mengengerüst vor. Die zur Bewertung herangezogenen Kriterien werden anschließend in Kapitel 5 genauer erläutert. Daraufhin trifft das sechste Kapitel eine Vorauswahl an OR-Mappern und definiert ein Bewertungsschema. Das restliche Kapitel zeigt die Ergebnisse der Evaluation, wobei jedes Werkzeug zunächst einzeln vorgestellt wird. In Kapitel 7 geben wir schließlich beruhend auf den Erkenntnissen aus der Evaluationsphase eine Empfehlung für die Cinovo AG ab.

2 Ablauf der Fachstudie

Nachfolgend wird der Ablauf dieser Arbeit vorgestellt, die in knapp sechs Monaten vom 01.06.2011 bis 24.11.2011 durchgeführt wurde.

2.1 Phasen

Beginn der Fachstudie

Die Fachstudie begann am 1. Juni 2011 mit einem Kick-Off-Meeting in den Geschäftsräumen der Cinovo AG in Stuttgart-Mitte. Nach einer kurzen Vorstellung der Cinovo AG schilderten uns zwei Vertreter des Unternehmens die momentane Situation und ihr Problem hinsichtlich der Auswahl eines geeigneten OR-Mappers. Da wir im Vorfeld Analysefragen vorbereitet hatten, konnten wir uns in der anschließenden Fragerunde ein erstes Bild über die Anforderungen an die von der Cinovo AG angestrebte Werkzeug-Lösung machen.

Projektplan und Recherche

Im Anschluss haben wir aus den Erkenntnissen des Kundengesprächs zum einen den Ablauf der Fachstudie geplant, zum anderen den Markt nach geeigneten OR-Mappern untersucht. Nachdem der erste Überblick verschaffen war, haben wir in einem ersten Schritt eine Liste der vielversprechenden OR-Mappern erstellt. Diese enthält zu jedem Werkzeug relevante Informationen wie beispielsweise dessen Lizenz und fasst Herstelleraussagen stichwortartig zusammen.

Analyse der Bewertungskriterien

In einem späteren Treffen wurden wichtige Bewertungskriterien mit unserem Ansprechpartner bei der Cinovo AG erörtert und festgehalten. Auch wurde die Relevanz der Kriterien angesprochen und diese auf einer Ordinalskala eingestuft. Darüber hinaus wurden konkrete Anforderungen für die Erfüllung des jeweiligen Kriteriums vereinbart. Es stellte sich bei der Diskussion heraus, dass einige Kriterien unter allen Umständen erfüllt sein müssten. Diese Kriterien wurden deshalb als K.O.-Kriterien eingestuft und bei der Bewertung als solche behandelt.

Voruntersuchung

In einer Voruntersuchung wurden alle gefundenen Werkzeuge auf die vereinbarten K.O.-Kriterien hin geprüft und mit Hilfe einer Tabelle diejenigen Werkzeuge bestimmt, die alle K.O.-Kriterien erfüllen. Durch dieses Vorgehen konnte im weiteren Verlauf der Fokus auf einen kleinen Teil der Werkzeuge (sog. Shortlist) gelegt werden.

Definition eines Bewertungsschemas

Um eine Vergleichsbasis zwischen den Werkzeugen zu schaffen, wurde von uns als Nächstes ein Bewertungsschema festgelegt, das auf einem einfachen, mathematischen Verfahren basiert und jedem untersuchten OR-Mapper eine Punktzahl sowie eine zugehörige Endnote zuordnet.

Evaluation der Werkzeuge

In der Evaluationsphase haben wir die in der Voruntersuchung ausgewählten Werkzeuge auf alle weiteren Bewertungskriterien überprüft und je nach Erfüllungsgrad des jeweiligen Kriteriums bewertet bzw. auf der Ordinalskala eingestuft. Daraus ließ sich eine Gesamtpunktzahl und schließlich eine Endnote für jedes Werkzeug berechnen.

Erarbeitung der Empfehlung

Da die Endnoten der OR-Mapper auf Basis eines mathematischen Verfahrens entstanden sind, wurde im letzten Schritt die Plausibilität der Ergebnisse überprüft. Dabei wurden Vor- und Nachteile der einzelnen OR-Mapper unter Berücksichtigung des Nutzungsszenarios des Industriepartners abgewägt. Auch der subjektive Eindruck floss bei der Plausibilitätsüberprüfung mit ein.

Abgabe und Ende der Fachstudie

Im November wurde die Fachstudie mit der Abgabe dieser Ausarbeitung und der Präsentation der Ergebnisse in einem Abschlussvortrag beendet.

2.2 Zeitlicher Verlauf

Das nachfolgende Gantt-Diagramm zeigt den zeitlichen Verlauf der zuvor beschriebenen Phasen und die erreichten Meilensteine.

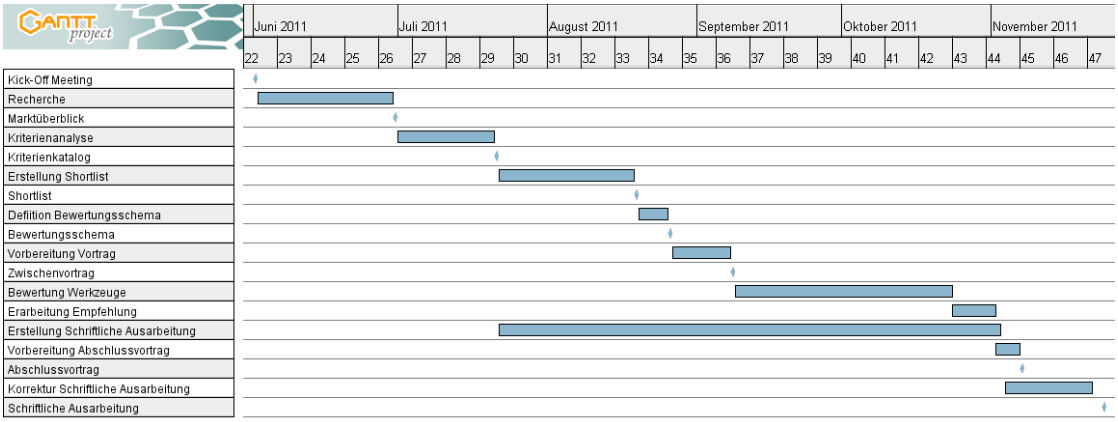


Abbildung 1: Verlauf der Fachstudie

3 Marktüberblick

Dieses Kapitel stellt einen Überblick über die aktuell verfügbaren OR-Mapping-Werkzeuge auf dem Markt dar. Jeder OR-Mapper wird stichwortartig vorgestellt. Alle Aussagen stammen von den Herstellern und können darum mehr versprechen, als der OR-Mapper tatsächlich leistet.

Active Objects

- ▷ Ausgelegt auf Einfachheit und leichte Bedienung
- ▷ Performanz ist kein primäres Ziel
- ▷ Leichte Integration in existierendes Produkt
- ▷ Einsatz des bestehenden Datenbankschemas möglich
- ▷ Automatische Generierung des Datenbankschemas
- ▷ Unterstützt Lazy-Loading und Caching-Mechanismen

Untersuchte Version
0.8.2 (22.04.08)

Lizenz
Apache 2.0

Webseite
activeobjects.java.net

Active JDBC

- ▷ Implementierung des Active-Record-Entwurfsmusters
- ▷ Inspiriert von ActiveRecord aus Ruby on Rails
- ▷ Paradigma: Konvention vor Konfiguration
- ▷ Leichte Bedienung und Leichtgewichtigkeit
- ▷ Reduzierung des Codes auf das Minimum
- ▷ Kenntnisse in Datenbanksprache SQL erforderlich

Untersuchte Version
1.2 (10.08.11)

Lizenz
Apache 2.0

Webseite
code.google.com/p/jdbc

Ammentos

- Auf Leichtgewichtigkeit und Einfachheit ausgelegt
- Verwendung von JDK5-Annotationen
- Keine Installation und Konfiguration erforderlich
- Keine Abhängigkeiten (Stand-Alone-Library)
- Hersteller bietet eine Support-Lizenz an

Untersuchte Version

1.3.7 (24.11.08)

Lizenz

GPL

Webseite

ammentos.org

Apache Cayenne

- Portabilität zwischen Datenbanken mit JDBC-Schnittstelle
- SQL-Kenntnisse sind nicht erforderlich
- Einfacher Code zur Validierung der Datenmodelle
- Lazy-Loading und Caching zur Performanzsteigerung
- GUI-Tool zur Modellierung des Datenbankschemas

Untersuchte Version

3.0.2 (21.06.11)

Lizenz

Apache 2.0

Webseite

cayenne.apache.org

Apache JDO

- Implementierung des JDO-Standards
- Transparente Persistenz der POJOs
- Unterstützt nicht-relationale Datenbanken
- Leichte Bedienung

Untersuchte Version

3.0 (08.04.10)

Lizenz

Apache 2.0

Webseite

db.apache.org/jdo

Apache ObjectRelationalBridge

- Unterstützung verschiedener Persistenz-APIs
- Einfache Bedienung und Leichtgewichtigkeit
- Einfache Integration in bestehenden Code
- Transparente Persistenz: Freie Vererbungshierarchie
- OR-Mapping mittels XML-Repository
- Mehrere Datenbanken einsetzbar

Untersuchte Version
1.0.4 (01.01.06)

Lizenz
Apache 2.0

Webseite
db.apache.org/ojb

Apache OpenJPA

- Implementierung des JPA-Standards
- Verwendung der Datenbanksprache JPQL
- Konfiguration über XML-Dateien
- Unterstützt Datenbanktransaktionen
- Früher Apache Kodo genannt

Untersuchte Version
2.1.1 (27.07.11)

Lizenz
Apache 2.0

Webseite
openjpa.apache.org

Apache Torque

- Zugriff auf User-Klassen über XML-Schemata anstatt Reflection
- Generierung der Modelle aus bestehender Datenbank möglich
- Verbirgt Implementierungsdetails der Datenbanken
- Ursprünglich Teil des Turbine-Frameworks

Untersuchte Version
3.3.1 (18.02.11)

Lizenz
Apache 2.0

Webseite
db.apache.org/torque

Athena Framework

- Bietet Unterstützung für Cloud-Applikationen an
- Auf hohe Usability ausgelegt
- Verwendung der Datenbanksprache EJBQL
- Konfiguration über XML-Dateien
- Unterstützt Multitenancy (Mandantenfähigkeit)
- Verfügt über eine GUI zur Änderung des Datenbankschemas
- Stellt Hilfsmittel zur Datenmigration bereit
- Kostenpflichtiger Support erhältlich

Untersuchte Version

2.0.0 (20.03.11)

Lizenz

LGPL

Webseite

athenasource.org

Carbonado

- Erweiterbar und hochperformant
- Verwendung von Java-Annotationen
- Unterstützung von nicht-SQL-konformen Datenbanken
- Definition der Datenmodelle über Schnittstellen oder abstrakte Klassen
- Keine externe Konfigurationsdateien
- Einsatz eines Repositories (Datenbank-Gateway)
- Gebrauch des Transaktionskonzepts
- Ursprünglich von Amazon entwickelt

Untersuchte Version

1.2.2 (10.12.10)

Lizenz

Apache 2.0

Webseite

carbonado.sf.net

Castor

- Verwendung einer XML-basierenden Mapping-Datei
- Code-Generierung anhand existierender XML-Schemen
- Implementierung eines Daten-Caches
- Verwendung der Datenbanksprache OQL
- Unterstützt Datenbanktransaktionen
- Integration in andere Frameworks möglich

Untersuchte Version
1.3.2 (29.03.11)

Lizenz
Apache 2.0

Webseite
castor.org

DataNucleus

- Implementiert offene Persistenzstandards wie JDO und JPA
- Features sind einzelne Plug-Ins
- Datenbankabfragen mit Hilfe verschiedener Abfragesprachen
- Unterstützt nicht-relationale Datenbanken
- Basiert auf OSGi-Technologie

Untersuchte Version
3.0.0 (01.08.11)

Lizenz
Apache 2.0

Webseite
datanucleus.org

Dozer

- Java Bean nach Java-Bean-Mapper
- Bereitstellung eines Eclipse-Plug-Ins
- Konfiguration über Property-Datei
- Verwendung von XML-Dateien sowie Annotationen

Untersuchte Version
5.3.2 (15.02.11)

Lizenz
Apache 2.0

Webseite
dozer.sf.net

Ebean

- Einfachere API als JPA-Standard
- Konfiguration über Property-Datei
- Verwendung von Java-Annotationen
- Kombination der JPA-Merkmale und „Relational“-Merkmale (z.B. aus IBatis)

Untersuchte Version
2.7.3 (23.03.11)

Lizenz
LGPL

Webseite
avaje.org

EclipseLink

- Referenzimplementierung des JPA-Standards
- Ausgelegt auf Vollständigkeit, Verständlichkeit und Vielseitigkeit
- Unterstützung von einigen Persistenzstandards (z.B. JPA, JAXB)
- Konfiguration über XML-Dateien
- Automatische Generierung des Datenbankschemas
- Verwendung von Caching-Mechanismen
- Interaktion mit verschiedenen Datensystemen
- Unterstützt Lazy-Loading
- Migration von anderen OR-Mapping-Werkzeugen

Untersuchte Version
2.2.1 (29.07.11)

Lizenz
Eclipse Public License

Webseite
eclipse.org/eclipselink

Enterprise Objects Framework (EOF)

- Bestandteil von WebObjects (Applikationsserver)
- GUI-Tool zum OR-Mapping
- Benutzung vorhandener Datenbankschemas
- Existenz diverser Open-Source-Implementierungen
- Vertrieb durch Apple

Untersuchte Version
5.2.3 (01.12.08)

Lizenz
Proprietär

Webseite
apple.com/webobjects

Floggy

- Für J2ME-/MIDP-Anwendungen entwickelt
- Fokus auf Performanz gelegt
- Leichtgewichtig (Entwicklung für mobile Geräte)
- Definition der Datenmodelle über Schnittstellen
- Unterstützt Lazy-Loading
- Stellt Hilfsmittel zur Datenmigration bereit
- Übersichtliche Dokumentation

Untersuchte Version

1.4.0 (09.02.11)

Lizenz

Apache 2.0

Webseite

floggy.sf.net

Hibernate

- Implementierung des JPA-Standards
- Integration in Applikationsserver und Servlet-Engines
- Abfragesprache: Hibernate Query Language (HQL)
- OR-Mapping mittels XML-Datei oder Annotationen
- Unterstützung von Vererbungsbeziehungen
- Kompatibilität mit verschiedenen Datenbanken
- Einsatz in mehr als 10.000 Java-Projekten

Untersuchte Version

3.6.6 (21.07.11)

Lizenz

LGPL

Webseite

hibernate.org

Java Ultra-Lite Persistence

- Leichtgewichtig (kleine Dateigröße, Stand-Alone)
- Unterstützung von Vererbung
- Unterstützt Lazy-Loading und Caching-Mechanismen

Untersuchte Version

3.0.1 (09.06.11)

Lizenz

GPL

Webseite

julp.sf.net

JPMapper

- Fokus auf Schnelligkeit und Flexibilität gelegt
- Einfache Bedienung
- SQL-Statements sind nicht erforderlich
- Implementiert das Facade-Entwurfsmuster
- Konfiguration über Property-Dateien

Untersuchte Version
0.7.1Beta (28.12.10)

Lizenz
LGPL

Webseite
jpmapper.sf.net

Oracle TopLink

- Implementierung des JPA-Standards 1.0 und 2.0
- Fokus auf hohe Performanz, Skalierbarkeit und Flexibilität gelegt
- Verwendet Komponenten von EclipseLink

Untersuchte Version
11.1.1.4.0 (01.11)

Lizenz
Oracle License

Webseite
oracle.com/.../toplink

ORMLite

- Speziell als leichtgewichtiges Werkzeug entwickelt
- Unterstützt die Android Mobile Plattform
- Konfiguration der Klassen mittels Java-Annotationen
- Flexibler Query-Builder wird bereitgestellt
- Unterstützt Datenbanktransaktionen
- Generierung von Datenbanktabellen

Untersuchte Version
4.25 (22.08.11)

Lizenz
Eigene OS-Lizenz

Webseite
ormlite.com

Persist

- ▷ Stellt minimale Mapping-Funktionalität bereit
- ▷ Auf hohe Performanz, einfache Benutzbarkeit und Integration ausgelegt
- ▷ Keine Konfigurationsdateien
- ▷ Definition des Datenmodells über Annotationen
- ▷ Verwendung der Datenbanksprache SQL
- ▷ Benötigt nahezu kein explizites Mapping

Untersuchte Version
1.1.1 (03.02.11)

Lizenz
BSD

Webseite
github.com/rufiao/persist

Prevayler

- ▷ Zeichnet sich durch Einfachheit, Schnelligkeit und Fehlertoleranz aus
- ▷ Objekte werden im Arbeitsspeicher gehalten
- ▷ Dump auf ein nichtflüchtiges Medium in regelmäßigen Abständen

Untersuchte Version
2.3 (08.06.06)

Lizenz
BSD

Webseite
prevayler.org

QuickDB

- ▷ Ausgelegt auf Einfachheit
- ▷ Wenig Konfigurationsaufwand
- ▷ Unterstützung von Vererbungshierarchien der Datenmodelle
- ▷ Generierung von Datenbanktabellen

Untersuchte Version
1.3-Beta2 (03.07.10)

Lizenz
LGPL

Webseite
code.google.com/quickdb

Siena

- Ausgelegt auf Leichtgewichtigkeit
- Einfache Bedienung
- Keine externen Abhängigkeiten
- Verwendung von Annotationen
- Unterstützung von nicht-relationalen Datenbanken
- Implementierung des Active-Record-Entwurfsmusters
- Als Modul im Play!-Framework verfügbar

Untersuchte Version

1.0.0-b5 (24.06.11)

Lizenz

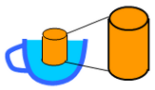
Apache 2.0

Webseite

sienaproject.com

SimpleORM

- Keine externen Abhängigkeiten
- Einfache Konfiguration
- Kein Gebrauch von Reflection, XML und Annotationen
- Vermeidung exotischer Technologien wie Byte Code Generation
- Ausgestattet mit einer einfachen Objektstruktur und Architektur



Untersuchte Version

3.11 (22.08.09)

Lizenz

Apache 1.1

Webseite

simpleorm.org

Speedo

- Open-Source-Implementierung des JDO-Standards 1.0.1
- Caching von persistenten Objekten
- Generierung des Datenbankschemas
- Abhängigkeiten zu einigen OW2-Frameworks

Untersuchte Version

1.4.5 (22.05.06)

Lizenz

LGPL

Webseite

speedo.ow2.org

4 Nutzungsszenario

Bei der Cinovo AG [Cin11] handelt es sich um ein Unternehmen, das in erster Linie Software für Börsenhändler entwickelt und vertreibt. Diese Software basiert auf einem Java-Applikationsserver und nutzt für die Verwaltung der Daten die vorhandene OR-Mapper-Bibliothek Hibernate. Da für den Betrieb der Anwendung leistungsstarke Server bzw. Workstations verwendet werden, wurde weniger Wert auf Leichtgewichtigkeit gelegt, sondern mehr auf die Funktionalität und die Möglichkeit, komplexe Strukturen darzustellen, geachtet.

Um die Kompetenzen des Unternehmens auszubauen, wird ein neuer Geschäftsbereich aufgebaut. Ziel des Geschäftsbereichs ist es, Dienstleistungen und Produkte für die Automobilindustrie zu vertreiben. Als erstes Projekt wird ein sogenannter Car-PC (s. Abbildung 2) entwickelt. Der Car-PC wird über einen Anschluss mit dem Auto verbunden und zeichnet während der gesamten Betriebszeit Daten auf. Nach dem Abschluss einer Fahrt können diese Daten auf einem anderen Rechner ausgewertet werden, um so Aufschluss über das Verhalten des Autos im Betrieb zu erhalten.

Bei den Daten handelt es sich um Messdaten, die das Auto zur Verfügung stellt und die mehrmals pro Sekunde in der Datenbank gespeichert werden sollen sowie um GPS-Koordinaten, die einmal pro Sekunde eingetragen werden. Da das Produkt auf dem Car-PC nur Daten in eine Datenbank schreibt und keine ausliest, hat dies starke Auswirkungen auf die Bewertungskriterien.

Zudem verfügt der verwendete Car-PC nur über eine eingeschränkte Leistungsfähigkeit. Daher eignet sich hier eine schwergewichtige und komplexe Lösung wie beim ersten Produkt der Cinovo AG nicht. Stattdessen wird eine leichtgewichtige Lösung bevorzugt, die mit den vorhandenen Ressourcen sparsam umgeht. Diese Einschränkung hat aber keine Auswirkungen auf die Software, die später zur Auswertung verwendet werden soll.

Da bei allen bisherigen Produkten der Cinovo AG das freie Datenbankmanagementsystem PostgreSQL zum Einsatz kommt und das Unternehmen mit diesem DBMS bislang gute Erfahrungen gemacht hat, möchte der Industriepartner auch bei der Entwicklung der Software für den Car-PC auf dieses DBMS setzen. Darum muss der gesuchte OR-Mapper in der Lage sein, mit PostgreSQL ohne Probleme umgehen zu können.

Abbildung 2: Abbild eines Car-PCs

4.1 Mengengerüst

Bei der Anwendung für den Car-PC kommen drei verschiedene Modelle zum Einsatz, um die zu messenden Daten des Autos abzuspeichern:

- Ein *Device* ist eine Komponente des Autos, die Daten an die Anwendung sendet.
- Ein *Measurement* ist ein solches Datenpaket.
- Ein *PositionFix* speichert Position, Geschwindigkeit und ähnliche Daten zu einem bestimmten Zeitpunkt ab.

In einem Auto gibt es etwa 40 *Device*-Objekte, die für die Anwendung relevant sind. Sie liefern pro Sekunde zwischen 1 und 5 Datenpakete, die in *Measurement*-Objekten abgespeichert werden. Zudem wird einmal pro Sekunde mit Hilfe eines GPS-Adapters ein *PositionFix*-Objekt erstellt.

5 Bewertungskriterien

Dieses Kapitel stellt die Kriterien vor, die mit dem Industriepartner abgesprochen wurden. Die Skala eines Kriteriums dient dazu, den Vergleich zwischen den verschiedenen Werkzeugen zu ermöglichen. Ein Werkzeug wird auf einer Stufe der Skala eingeordnet, wenn es die Anforderung des Industriepartners für diese Stufe erfüllt.

5.1 K.O.-Kriterien

Das gesuchte Werkzeug muss wichtige Anforderungen unbedingt erfüllen. Diese spiegeln sich in den nachfolgenden K.O.-Kriterien wieder.

5.1.1 Lizenzierung

Beschreibung

Dieses K.O.-Kriterium beschäftigt sich damit, unter welcher Lizenz das zu bewertende Werkzeug zur Verfügung gestellt wurde.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Das Werkzeug wurde unter einer Open-Source-Lizenz veröffentlicht.
- ▷ Die Lizenz erlaubt eine kommerzielle Nutzung.
- ▷ Die Lizenz enthält keine starke Copyleft-Klausel.

Beispiele

- ▷ Apache 2.0 von der Apache Software Foundation
- ▷ LGPL von der Free Software Foundation

5.1.2 Leichtgewichtigkeit/Abhängigkeiten

Beschreibung

In diesem K.O.-Kriterium geht es darum, dass das zu bewertende Werkzeug leichtgewichtig sein muss (s. 1.1).

Skala

$\oplus$ / $\ominus$

Anforderung des Industriepartners

- ▷ Es dürfen maximal 5 externe Bibliotheken vom Werkzeug eingebunden werden. Der JDBC-Treiber wird hierbei nicht berücksichtigt.

5.1.3 Status der Entwicklung

Beschreibung

Alle Werkzeuge, die aktiv weiterentwickelt werden, sich offiziell nicht mehr in der Testphase befinden und bereits seit längerer Zeit auf dem Markt sind, genügen diesem K.O.-Kriterium.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Das letzte stabile Release des Werkzeugs liegt nicht länger als 9 Monate zurück.
- ▷ Die letzte Entwicklungsaktivität (HEAD-Revision im Repository) ist nicht länger als 4 Monate her.
- ▷ Es ist ersichtlich, dass die Weiterentwicklung nicht eingestellt wurde bzw. in kurzer Zeit eingestellt wird.
- ▷ Das Werkzeug befindet sich nach Herstellerangaben nicht mehr in der Testphase.
- ▷ Der Startpunkt der Entwicklung (erste Revision im Repository) ist länger als 12 Monate her.

5.1.4 Unterstützung des Datenbankmanagementsystems

Beschreibung

Das zu bewertende Werkzeug muss nativ das zu verwendende Datenbankmanagementsystem (DBMS) unterstützen.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Das Werkzeug bietet nach Herstellerangaben Unterstützung für PostgreSQL an.
- ▷ Die Verbindung zu PostgreSQL wurde vom Hersteller getestet und ist von uns als stabil einzustufen.

5.2 Weitere relevante Kriterien

Zur Beurteilung der Werkzeuge tragen maßgeblich weitere Kriterien bei. Diese werden absteigend nach ihrer Relevanz in diesem Abschnitt charakterisiert.

5.2.1 Performanz der INSERT-Abfragen

Beschreibung

Das Werkzeug muss Daten performant in die Datenbank schreiben können, sodass die maximale Last des Mengengerüsts (s. 4.1) verarbeitet werden kann.

Für den Lasttest haben wir ein Skript entwickelt, das einen zehnminütigen Testlauf auf einem Car-PC durchführt. Das Skript ermittelt zunächst in einem Brute-Force-Verfahren die maximale Anzahl der INSERT-Befehle und führt anschließend basierend auf dem Mengengerüst ein Testlauf durch, der die realistischen Bedingungen besser simuliert. Diese Simulation bezieht sich auf die maximale Last: Alle 200 Millisekunden müssen 40 *Measurement*-Objekte sowie jede Sekunde ein *PositionFix*-Objekt gespeichert werden. Hochgerechnet müssen demnach 120.000 *Measurement*-Instanzen und 600 *PositionFix*-Objekte in 10 Minuten persistiert werden. Falls das Brute-Force-Verfahren deutlich mehr Datensätze in dieser Zeit speichert, als es das Mengengerüst vorschreibt, kann sich das Werkzeug während der Durchführung des Lasttests auch im Leerlauf befinden.

Skala

$\oplus$ / $\odot$ / $\ominus$

Anforderungen des Industriepartners

- $\oplus$ Das Werkzeug verbringt während der Durchführung des Performanztests 20 oder mehr Prozent der Zeit im Leerlauf und kann mühelos mit dem Mengengerüst umgehen. Dies zeigt sich auch daran, dass im Brute-Force-Verfahren deutlich mehr Objekte gespeichert werden können, als es das Mengengerüst fordert.
- $\odot$ Das Werkzeug arbeitet das Mengengerüst in der vorgegebenen Zeit zuverlässig ab, d.h. alle zu speichernden Datensätze wurden persistiert und es wurden nur vereinzelnd Durchläufe nicht in der vorgeschriebenen Zeit erledigt.
- $\ominus$ Das Werkzeug kann mit der maximalen Last nicht umgehen. Dies zeigt sich darin, dass mehr als 10 Prozent der Durchläufe nicht in dem vorgegebenen Zeitintervall (200ms) erledigt werden konnten.

5.2.2 Dokumentation/Support

Beschreibung

Es ist gefordert, dass die Dokumentation des Werkzeugs zumindest aus einem „Getting Started“-Dokument besteht, damit sich der Benutzer in kurzer Zeit einen Überblick über den Funktionsumfang des Werkzeugs machen kann und sich nicht langwierig durch eine

API-Dokumentation oder im schlimmsten Fall durch den Quellcode selbst kämpfen muss. Support durch den Hersteller muss nicht gewährleistet sein.

Skala

⊕ / ⊙ / ⊖

Anforderungen des Industriepartners

- ⊕ Es existiert eine ausführliche Dokumentation, die maßgeblich über ein „Getting Started"-Dokument hinausgeht.
- ⊙ Der Hersteller stellt mindestens ein für die Einarbeitung hilfreiches „Getting Started"-Dokument bereit.
- ⊖ In allen anderen Fällen.

5.2.3 Simplizität des Abspeicherns

Beschreibung

Das Abspeichern der Modelle sollte möglichst einfach gestaltet sein. Darunter fällt unter anderem, ob mehr als ein Funktionsaufruf benötigt wird, um das Modell zu speichern bzw. mehr als ein Persistenzmanager verwendet wird. Ein Peristenzmanager ist eine Komponente, die persistente Objekte verwaltet und Datenbankoperationen durchführt.

Skala

⊕ / ⊙ / ⊖

Anforderungen des Industriepartners

- ⊕ Falls ein Persistenzmanager verwendet wird, so sollte die Speicherung mit einem einzigen Manager möglich sein. Wird ein solcher nicht verwendet, muss das Abspeichern in einem selbsterklärenden Aufruf erfolgen.
- ⊙ Das Abspeichern ist mit Hilfe eines einzigen Aufrufes möglich.
- ⊖ Der Aufruf zum Abspeichern besteht aus mehreren Operationen.

5.2.4 Generierung des Datenbankschemas

Beschreibung

Aus der Definition der Modelle sollte sich das Datenbankschema für das zu verwendende DBMS generieren lassen.

Skala

⊕ / ⊖

Anforderungen des Industriepartners

- ▷ Das Datenbankschema kann auf Basis der Modelldefinitionen auf der Konsole oder in einer eigenen Datei im SQL-Format ausgegeben werden.

5.2.5 Anlegen von Datenbanken

Beschreibung

Falls vor dem eigentlichen Start des Produkts zunächst eine neue Datenbank mitsamt des Datenbankschemas erstellt werden muss, sollte das Aufsetzen der Grundstruktur der Datenbank vom Werkzeug selbst automatisch durchgeführt werden.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Das Werkzeug kann beim Start mit einer leeren Datenbank alle Tabellen selbst anlegen.

5.2.6 Einarbeitungsaufwand

Beschreibung

Es wird verlangt, dass das Werkzeug keinen langwierigen Einarbeitungsaufwand erfordert. Der Einarbeitungsaufwand wird dadurch gemessen, wie lange ein Entwickler benötigt, um mit dem Werkzeug ohne größere Probleme umgehen zu können. Hierbei ist unser subjektive Eindruck, der beim Implementieren eines Beispieles entsteht, ausschlaggebend.

Skala

$\oplus$ / $\odot$ / $\ominus$

Anforderungen des Industriepartners

- $\oplus$ Die Einarbeitung entspricht der Komplexität des Problems.
- $\odot$ Die Einarbeitung ist mit kleineren Problemen verbunden.
- $\ominus$ Während der Einarbeitung kommt es zu größeren Problemen.

5.2.7 Form der Modelldefinition

Beschreibung

Es sollte möglich sein, die Datenbankmodelle im Java-Code in Form von Annotationen zu definieren.

Skala

$\oplus$ / $\odot$ / $\ominus$

Anforderungen des Industriepartners

- ⊕ Die Datenbankmodelle können über Java-Annotationen beschrieben werden.
- ⊖ Die Datenbankmodelle können lediglich über XML-Dateien oder vergleichbaren Konfigurationsmöglichkeiten spezifiziert werden.

5.2.8 Community

Beschreibung

Spezifische Fragen bezüglich des Werkzeugs sollen von einer Internet-Community beantwortbar sein. Als Teil der Community gelten hierbei Foren, Mailing-Listen und Chats wie z.B. ein IRC-Kanal oder direkte Kontaktmöglichkeiten zu den Entwicklern.

Skala

⊕ / ⊙ / ⊖

Anforderungen des Industriepartners

- ⊕ Die Community ist sehr aktiv und beantwortet Fragen schnell und zuverlässig.
- ⊙ Es existiert eine Community, die Fragen über das Werkzeug sporadisch beantwortet.
- ⊖ Für das Werkzeug ist keine aktive Community vorhanden.

5.2.9 Vielfältiger Einsatz

Beschreibung

Da das Werkzeug unter Umständen auch nach dem Ablauf der Fachstudie für andere Softwareprojekte eingesetzt werden soll, ist es von entscheidendem Vorteil, wenn das Werkzeug nicht zu sehr auf das zu entwickelnde Produkt zugeschnitten ist.

Skala

⊕ / ⊙ / ⊖

Anforderungen des Industriepartners

- ⊕ Das Werkzeug ist modular aufgebaut und demnach erweiterbar.
- ⊙ Es werden SELECT-Befehle sowie weitere DBMS unterstützt.
- ⊖ In allen anderen Fällen.

5.2.10 Syntax von Abfragen

Beschreibung

Der Syntax von Datenbankabfragen sollte nicht ausschließlich von SQL oder einer anderen vergleichbaren Datenbanksprache vorgegeben werden. Das Werkzeug soll eine Möglichkeit vorsehen, die Syntax von Abfragen auch oder nur über Klassen und deren Funktionen bilden zu können.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Es werden zur Erstellung von Abfragen Funktionen bzw. Klassen zur Verfügung gestellt, sodass der Nutzer kein SQL oder ähnliche Datenbanksprachen verwenden muss.

5.2.11 Umgang mit INSERT- und UPDATE-Abfragen

Beschreibung

Es soll vom zu bewertenden Werkzeug keine Unterscheidung zwischen dem Speichern eines neuen Datenbankeintrags mittels einer INSERT-Abfrage und dem Ändern eines alten Datenbankeintrags mittels einer UPDATE-Abfrage im Java-Programmcode gemacht werden.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Zum Einfügen und Ändern von Datenbankeinträgen wird derselbe Befehl verwendet.

5.2.12 Vererbungsstrukturen zwischen Datenmodellen

Beschreibung

Das zu bewertende Werkzeug soll in der Lage sein, mit Vererbungsstrukturen zwischen Datenmodellen umzugehen.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Ein konzipiertes Codebeispiel, bei dem ein Datenmodell von einem anderen erbt, lässt sich mit Hilfe des Werkzeugs auf die relationale Datenbank abbilden.

5.2.13 Transaktionen

Beschreibung

Die Sicherstellung der Datenintegrität durch Transaktionen ist ein Feature heutiger Datenbanken und sollte vom Werkzeug unterstützt werden.

Skala

$\oplus$ / $\odot$ / $\ominus$

Anforderungen des Industriepartners

- $\oplus$ Transaktionen sind möglich, müssen jedoch nicht genutzt werden.
- $\odot$ Transaktionen sind möglich und müssen genutzt werden.
- $\ominus$ Transaktionen werden nicht unterstützt.

5.2.14 Antwort bei Fehlschlägen

Beschreibung

Falls es beim Speichern eines Datensatzes zu einem Fehler kommt, sollte vom Werkzeug anstelle einer Exception ein Boolean-Wert zurückgeliefert werden.

Skala

$\oplus$ / $\odot$ / $\ominus$

Anforderungen des Industriepartners

- $\oplus$ Das Rückgabeverhalten beim Auftreten eines Fehlers ist einstellbar.
- $\odot$ Ein Rückgabewert meldet fehlerhafte Datenbankzugriffe.
- $\ominus$ Fehler bei Datenbankzugriffen werden dem Aufrufer mittels Exceptions mitgeteilt.

5.2.15 Entity-Manager

Beschreibung

Das Werkzeug soll nicht zwangsweise fordern, jedes Datenmodell von einer Basisklasse ableiten zu müssen.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Es kann ein Entity-Manager verwendet werden, um selbst noch Ableitungen im Datenmodell einsetzen zu können.

5.2.16 Unterstützung bei Schemamigration

Beschreibung

Bei Änderungen im Datenmodell ist eine Unterstützung seitens des Werkzeugs bei der Migration des bestehenden Datenbankschemas hilfreich.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Das Werkzeug unterstützt den Entwickler bei der Schemamigration mit Hilfe einer technischen Vorrichtung.

5.2.17 Unterstützung bei Datenmigration

Beschreibung

Eine Unterstützung seitens des zu bewertenden Werkzeugs bei der Migration von Daten auf ein neues Datenbankschema ist ein hilfreiches Feature, das das Werkzeug anbieten sollte.

Skala

$\oplus$ / $\ominus$

Anforderungen des Industriepartners

- ▷ Das Werkzeug unterstützt den Entwickler bei der Datenmigration mit Hilfe einer technischen Vorrichtung.

5.3 Irrelevante Kriterien

Einige von uns vorgeschlagenen Kriterien spielten für den Industriepartner keine Rolle. Diese irrelevanten Kriterien sind der Vollständigkeit halber hier aufgelistet.

- ▷ **Visuelle Tools**: GUI-Tools, die Hilfe beim Erstellen des Datenbankschemas leisten, sind nicht von Bedeutung, da das Schema über die Datenmodelle definiert wird.
- ▷ **Extraktion des Schemas**: Die automatische Erstellung von Modellen aus einem existierenden Datenbankschema ist nicht erforderlich.
- ▷ **Validierung**: Die Validierung der Daten eines Modells ist nicht notwendig.
- ▷ **Lazy-Loading**: Das dynamische Nachladen von Daten beziehungsweise Verknüpfungen ist nicht gefordert.
- ▷ **Caching:** Da Daten direkt in die Datenbank gespeichert werden sollen, um Datenverluste zu vermeiden, wird kein Caching benötigt.

6 Evaluation

6.1 Voruntersuchung

Vor der eigentlichen Evaluationsphase fand eine Voruntersuchung statt, in der diejenigen Werkzeuge aussortiert wurden, die von vorneherein nicht den Wünschen des Industriepartners entsprachen. Die gekürzte Liste (Shortlist) enthält darum nur noch Werkzeuge, die alle K.O.-Kriterien erfüllen.

Die Vorauswahl geschah mit Hilfe einer Matrix (Tabelle 1 auf der nächsten Seite). Sie stellt dar, welche K.O.-Kriterien von den Werkzeugen erfüllt wurden. Bei einigen Werkzeugen mussten K.O.-Kriterien nicht näher untersucht werden, da bereits andere K.O.-Kriterien vom Werkzeug nicht erfüllt wurden.

Erklärung der Legende

Erfüllt ($\oplus$)	Alle Anforderungen des Kriteriums sind erfüllt.
Nicht erfüllt ($\ominus$)	Mindestens eine Anforderung des Kriteriums ist nicht erfüllt.
Nicht überprüft (?)	Das Kriterium wurde im Rahmen der Fachstudie nicht überprüft.

Erklärung der Kürzel

LIZ	Lizenz des Werkzeugs
DEP	Anzahl der externen Abhängigkeiten
REL	Letztes stabiles Release
ACT	Letzte Entwicklungsaktivität
TES	Werkzeug in Testphase
STA	Startpunkt des Projekts
DBMS	Unterstützung des Datenbankmanagementsystems

Name	LIZ	DEP	Status der Entwicklung				DBMS
			REL	ACT	TES	STA	
Active Objects	⊕	?	⊖	⊖	?	?	?
Active JDBC	⊕	⊕	⊕	⊕	⊖	⊕	⊕
Ammentos	⊖	?	⊖	?	?	?	?
Apache Cayenne	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Apache JDO	⊕	⊕	⊖	?	⊕	?	⊖
Apache ORB	⊕	?	⊖	?	?	?	?
Apache OpenJPA	⊕	⊖	⊕	⊕	⊕	?	⊕
Apache Torque	⊕	?	⊖	?	?	?	?
Athena	⊕	⊖	⊕	?	⊕	?	⊖
Carbonado	⊕	⊕	⊕	⊕	⊕	⊕	⊖
Castor	⊕	⊖	⊕	⊕	⊕	⊕	⊕
DataNucleus	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Dozer	⊕	⊕	⊕	⊕	⊕	⊕	⊖
Ebean	⊕	⊕	⊕	⊕	⊕	⊕	⊖
EclipseLink	⊕	⊖	⊕	⊕	⊕	?	⊕
EOF	?	?	⊖	?	?	?	?
Floggy	⊕	⊕	⊕	⊖	⊕	⊕	⊖
Hibernate	⊕	⊖	⊕	⊕	⊕	?	⊕
Java Ultra-Lite	⊖	?	⊕	?	?	?	?
JPMapper	⊕	?	⊕	?	⊖	?	?
Oracle TopLink	⊖	⊖	⊕	?	⊕	?	⊕
ORMLite	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Persist	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Prevayler	⊕	?	⊖	?	?	?	?
QuickDB	⊕	?	⊖	⊖	?	?	?
Siena	⊕	⊕	⊕	⊕	⊕	⊕	⊕
SimpleORM	⊕	⊕	⊖	⊕	⊕	⊕	⊕
Speedo	⊕	?	⊖	?	?	?	?

Legende Erfüllt: ⊕ | Nicht erfüllt: ⊖ | Nicht überprüft: ?

Tabelle 1: Shortlist aus der Voruntersuchung

6.2 Bewertungsschema

Grundsätzliches Vorgehen für ein Werkzeug

Sind alle Kriterien eines Werkzeugs bewertet, werden die Punktzahlen mit ihrer Gewichtung multipliziert und daraufhin aufsummiert. Die Summe wird durch die Gesamtanzahl der möglichen Punkten dividiert und das Werkzeug bekommt schließlich anhand des Quotienten eine Endnote zugewiesen.

Mathematisch gesprochen ist der Quotient Q_W für ein Werkzeug W definiert durch

$$Q_W = \frac{1}{P_g} \sum_i g_i p_i$$

Hierbei ist

- i ein Kriterium aus Kapitel 5.2,
- g_i die Gewichtung des Kriteriums i nach Absprache mit dem Industriepartner,
- p_i die Punktzahl des Kriteriums i anhand der Evaluationsergebnisse,
- P_g die maximal erreichbare Punktzahl.

Die Punktzahlen, Gewichtungen und Endnoten können mit Hilfe der nachfolgenden Tabellen ermittelt werden.

Punktzahlen

Ein Werkzeug wird auf einer Stufe der Skala eingeordnet, wenn es die Anforderung des Industriepartners für diese Stufe erfüllt und kann eine Punktzahl zwischen 0 und 2 Punkten für dieses Kriterium bekommen.

Ergebnis	Punktzahl	Bedeutung im Allgemeinen
$\oplus$	2 Punkte	Die Anforderung wird vollständig erfüllt.
$\odot$	1 Punkt	Die Anforderung wird teilweise erfüllt.
$\ominus$	0 Punkte	Das Werkzeug wird der Anforderung nicht gerecht.

Gewichtungen

Alle möglichen Gewichtungen befinden sich auf der Ordinalskala $[0, 5]$. Die Gewichtung $g_i = 0$ repräsentiert die irrelevanten Kriterien, die für die Evaluation keine Rolle spielen.

Kriterium i	g_i
Performanz der INSERT-Abfragen	5
Dokumentation/Support	5
Simplizität des Abspeicherns	5
Generierung des Datenbankschemas	5
Anlegen von Datenbanken	5
Einarbeitungsaufwand	4
Form der Modelldefinition	4
Community	4
Vielfältiger Einsatz	4
Syntax von Abfragen	4
Umgang mit INSERT- und UPDATE-Abfragen	4
Vererbungsstrukturen zwischen Datenmodellen	4
Transaktionen	3
Antwort bei Fehlschlägen	3
Entity-Manager	2
Unterstützung bei Schemamigration	1
Unterstützung bei Datenmigration	1

Tabelle 2: Gewichtung der Bewertungskriterien

Endnoten

Aus den maximal erreichbaren Punkten und den Gewichtungen ergibt sich eine Gesamtpunktzahl P_g von 126 Punkten.

Q_W [%]	**Benötigte Punkte**	**Endnote**
100-95	119	1+
95-90	113	1
90-85	107	1-
85-80	100	2+
80-75	94	2
75-70	88	2-
70-65	81	3+
65-60	75	3
60-55	69	3-
55-50	63	4+
50-45	56	4
45-0	0	5

Tabelle 3: Endnote anhand der erreichten Punktzahl

6.3 Werkzeuge

6.3.1 Apache Cayenne

Apache Cayenne [Cay11] bewirbt sich unter anderem damit, dass für die Verwendung keine XML- oder Annotation-basierte Konfiguration des Werkzeugs notwendig ist. Dies trifft nach eingehender Begutachtung jedoch nur zu, falls die mitgelieferte GUI-Anwendung zur Konfiguration und Modellierung verwendet wird. Will man auf eine GUI-basierte Modellierung verzichten, ist man dazu gezwungen, die Modellierung in XML-Form anzugeben. Das Werkzeug ist auch allgemein stark auf die Benutzung der GUI-Anwendung ausgelegt und bietet viele Funktionen ausschließlich in selbiger an.

Untersuchte Version

3.0.2 (21.06.11)

Lizenz

Apache 2.0

Webseite

cayenne.apache.org

```
<data-map [...]>
    <db-entity name="device" schema="public">
        <db-attribute name="description" type="VARCHAR" length="256"/>
        <db-attribute name="id" type="INTEGER" isPrimaryKey="true"
                isMandatory="true" length="10"/>
        <db-attribute name="name" type="VARCHAR" length="256"/>
        <db-attribute name="type" type="VARCHAR" length="256"/>
        <db-attribute name="urn" type="VARCHAR" length="256"/>
    </db-entity>
</data-map>
```

Listing 1: Auszug aus der XML-Definition des *Device*-Modells

Die Dokumentation von Apache Cayenne ist ausführlich und nach etwas Einarbeitungszeit auch verständlich, jedoch existieren zum Teil noch Sektionen, die als „Draft“ gekennzeichnet sind. Es gibt nicht nur ein „Getting Started“-Tutorial und eine JavaDoc-API, sondern auch weitere Informationen zu einzelnen Teilen des Werkzeugs sowie ein Leitfaden zur Einarbeitung in den Cayenne Modeler und die Erweiterung für Remote Clients. Da das Werkzeug allerdings recht umfangreich ist und man sich sowohl in die Modelling-UI als auch in die Verwendung von Apache Cayenne an sich einarbeiten muss, kann es bei der Einarbeitung durchaus zu kleineren Problemen kommen. Sollte versucht werden, auf den Cayenne Modeler zu verzichten, wäre dies nicht ohne erheblichen Einarbeitungsaufwand möglich.

Die Community von Apache Cayenne besteht im Wesentlichen aus einer Mailing-Liste, die anfallende Fragen in der Regel innerhalb weniger Tage beantwortet und bei der Lösung auftretender Probleme hilft. Weitere vom Entwickler unterstützte Foren oder Chats konnten nicht gefunden werden, jedoch bieten zwei der Hauptentwickler von Apache Cayenne einen kommerziellen Support an, der von persönlicher Beratung bei der Erstellung von Projekten über das Entwickeln von kundenspezifischen Features bis hin zu direktem

Kontakt mit den Entwicklern zur Fehlerbehebung reicht.
Datenbankabfragen werden in Apache Cayenne gänzlich ohne SQL-Syntax formuliert, was es auch Entwicklern ohne SQL-Kenntnisse erlaubt, effizient mit dem Werkzeug umzugehen. Über die Persistenzklassen, die von dem mitgelieferten Cayenne Modeler generiert werden können, werden die Änderungen an einem Objekt an den zugehörigen Persistenzmanager übergeben, der diese dann an die Datenbank übermittelt. Hierbei werden neue, gelöschte und geänderte Objekte gleich behandelt, was für eine übersichtlichere Strukturierung der Query-Syntax sorgt. Auch SELECT-Abfragen müssen nicht in SQL-Syntax formuliert werden, sondern können über Funktionen, die auf einem der Datenmodelle basieren, durchgeführt werden.
Speziell für SELECT-Abfragen verfügt Cayenne über einige performanzsteigernde Optimierungen. Hierzu gehören Caching, Lazy-Loading und Prefetching von relationalen Daten. Prefetching ermöglicht es im Gegensatz zum Lazy-Loading, durch einen einzigen Query-Aufruf mehr als nur einen Objekttyp aus der Datenbank zu lesen. Das hat den Vorteil, dass auf der Datenbank intern erheblich wenigere Abfragen ausgeführt werden müssen.

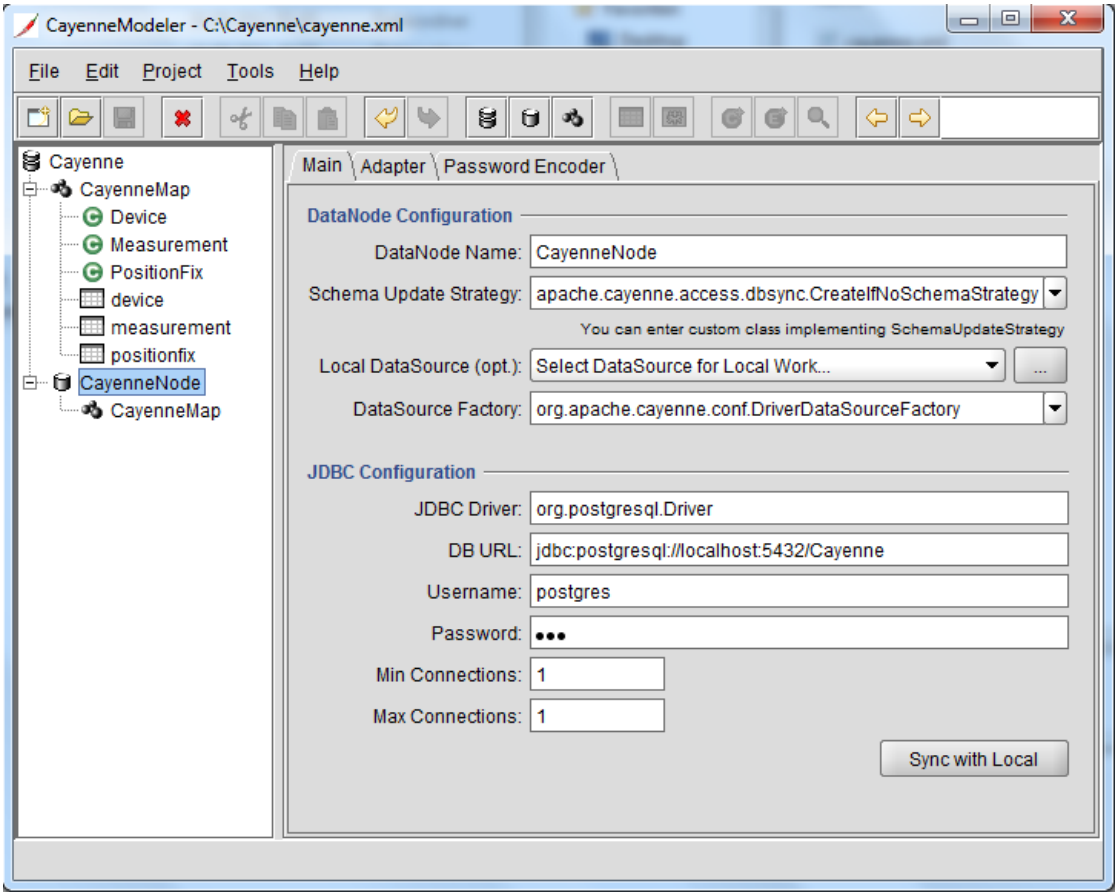


Abbildung 3: Cayenne Modeler

Datenmodelle, die von Apache Cayenne persistiert werden sollen, müssen alle von der internen Klasse *CayenneDataObject* abgeleitet werden. Die daraus resultierende Klassenstruktur ist nicht sehr übersichtlich, besonders, da die Dokumentation dringend davon abrät, vom Cayenne Modeler erzeugte Klassen zu verändern, da diese bei erneuter Generierung gänzlich überschrieben werden. Dies erfordert, dass die bereits abgeleiteten Klassen ein weiteres Mal abgeleitet werden müssen.

Die GUI-Anwendung von Apache Cayenne, namentlich Cayenne Modeler, stellt Möglichkeiten zur Modellierung des Datenbankschemas, Konfiguration des Werkzeugs, Migration von vorhandenen Datenbankschemata sowie zur Generierung der Persistenzklassen zur Verfügung. Durch die für den Cayenne Modeler separat existierende Dokumentation ist die Einarbeitung in den Cayenne Modeler einfach und die angebotenen Funktionen sind nach kurzer Zeit verständlich.

Die Verwaltung von Datenbankschemata ist dank des Cayenne Modelers simpel gehalten. So kann über wenige Mausklicks ein bestehendes Datenbankschema nachgebaut und als Datenmodell verwendet werden. Bei Änderungen am Datenmodell oder am Datenbankschema bietet Apache Cayenne eine Migrationshilfe an. Der Nutzer kann dabei entscheiden, welche Änderungen auf Datenmodell und Datenbankschema angewandt werden sollen. Zudem stellt Apache Cayenne mehrere Möglichkeiten zur Verfügung, das erstellte Datenmodell als Schema auf die Datenbank abzubilden. Zum einen kann der Nutzer sich im Cayenne Modeler die SQL-Befehle zur Erstellung des Schemas anzeigen lassen, sie als Datei speichern oder direkt ausführen lassen. Zum anderen kann er eine Schema-Update-Strategie wählen, die das Schema automatisch zur Laufzeit erstellt, falls die Datenbank kein Schema enthält.

In Bezug auf die Leichtgewichtigkeit liegt Apache Cayenne mit drei externen Abhängigkeiten ohne JDBC im Mittelfeld der untersuchten Werkzeuge. Mit diesen Abhängigkeiten lässt sich auch die volle Funktionalität des Werkzeugs erreichen.

PRO	**CONTRA**
⊕ Simple Syntax	⊖ Mapping über XML-Dateien
⊕ Unterstützung bei der Migration des Schemas	⊖ Vererbung von Basisklasse an Datenmodelle erforderlich

6.3.2 DataNucleus

Untersuchte Version

3.0.0 (01.08.11)

Lizenz

Apache 2.0

Webseite

datanucleus.org

Als einziger OR-Mapper in der näheren Auswahl implementiert DataNucleus [Dat11] die offenen Persistenzstandards Java Data Objects (JDO) und Java Persistence API (JPA). Dies ermöglicht es, das Werkzeug bei Bedarf ohne große Probleme gegen ein anderes Werkzeug auszutauschen. Kandidaten, die mindestens einen der genannten Standards implementieren und daher als Ersatz für DataNucleus fungieren können sind beispielsweise OpenJPA, EclipseLink oder auch Hibernate, die jedoch alle drei nicht den Anspruch auf einen leichtgewichtigen Mapper haben und folglich im Rahmen der Voruntersuchung aus der näheren Auswahl entfernt wurden.

Aufgrund der Unterstützung dieser und weiterer Standards ist es auch das Werkzeug, das die meisten externen Abhängigkeiten benötigt und kann nur dann als leichtgewichtig im Sinne dieser Fachstudie bezeichnet werden, wenn der Entwickler zum Verwalten seiner Datenobjekte ausschließlich den JDO-Standard verwendet. Aus diesem Grund wurde im Rahmen der Fachstudie die Implementierung des JPA-Standards von DataNucleus nicht weiter untersucht, da dieser laut Herstellerangaben auf den JDO-Standard aufbaut und deshalb sowohl dessen Referenzen zusätzlich zu den eigenen Abhängigkeiten benötigt als auch einen Persistenzmanager vorschreibt.

DataNucleus basiert auf der OSGi-Technologie und ist daher modular aufgebaut. Dem Werkzeug lassen sich so auf einfache Weise weitere Features als Modul bzw. Plug-In hinzufügen. Diese können vom Entwickler selbst programmiert werden oder aus einer großen Auswahl entnommen werden.

Zur Erstellung des Schemas bietet DataNucleus ein externes Programm namens SchemaTool sowie einen eigenen Byte-Code-Enhancer an, die beide durch eine weitere JAR-Datei in die Anwendung eingebunden werden können. Der Byte-Code-Enhancer nimmt dabei die Datenmodelle in Form von Java-Klassen als Eingabe und produziert daraus vorkompilierte Objekte, die von DataNucleus persistiert werden können. Da diese Umwandlung je nach Anzahl der zu kompilierenden Java-Klassen lange dauern könnte, ist es empfehlenswert, die Konvertierung als Pre-Build-Schritt vor der eigentlichen Ausführung der Software durchführen zu lassen. Das SchemaTool kann anhand der Datenmodelle das Schema der angegeben Datenbank erstellen oder auch wieder aus der Datenbank löschen. Hilfreich kann hierbei die Validate-Funktion von SchemaTool sein, die ein existierendes Schema auf Kompatibilität mit dem derzeitigen Datenmodell prüft und Abweichungen feststellt. Des Weiteren kann mit dem SchemaTool das zu erstellende Datenbankschema im SQL-Format in einer Datei ablegt werden.

Falls die Entwicklung mit Hilfe von Eclipse erfolgt, bietet DataNucleus hierfür ein

Eclipse-Plug-In an, das dem Entwickler manche Arbeiten im Umgang mit DataNucleus abnimmt. So sind der Enhancer wie auch das SchemaTool direkt in die Oberfläche von Eclipse integriert und können ohne viel Konfigurationsaufwand auf ein bestehendes Java-Projekt angewandt werden. Alternativ können die Build-Systeme Ant oder Maven zur Konfiguration von DataNucleus eingesetzt werden, falls der Entwickler auf eine Java-IDE verzichtet, die das Erstellen der Software übernimmt.

Hervorzuheben ist die ausführliche Dokumentation, die auf den Webseiten des Werkzeugs zu finden ist. So bietet das „Getting Started“-Dokument einen ersten Überblick über die Funktionalitäten des Werkzeugs. Für die weiteren Einzelheiten wird der Leser je nach Wahl des Persistenzstandards auf ein weiterführendes Tutorial verwiesen, das Schritt für Schritt mit Hilfe eines konkreten Beispiels die Grundfunktionen des Werkzeugs beschreibt. Darüber hinaus hat das Werkzeug eine aktive Community, die im Wesentlichen aus einem Forum besteht, das DataNucleus-spezifische Fragen in den meisten Fällen zügig beantwortet. Falls diese Hilfe nicht ausreichen sollte, können je nach Bedarf diverse Supportverträge mit den Entwicklern abgeschlossen werden. Es ist außerdem möglich, DataNucleus-Schulungen gegen Entgelt zu besuchen.

Das Werkzeug kann individuell konfiguriert werden. Die Konfiguration kann entweder direkt im Java-Code erfolgen oder über eine Properties-Datei angegeben werden. Beispielsweise ist es möglich, das Caching der Objekte zu verbieten, bevor diese in der Datenbank abgespeichert werden. Über eine Einstellung lässt sich zudem das Schema automatisch beim Start des Programms erstellen. Das Mapping an sich kann über Java-Annotationen oder wahlweise auch über eine XML-Datei, die über eine Property dem Werkzeug mitgeteilt wird, erfolgen.

In der JDO-Implementierung von DataNucleus ist es vom Hersteller empfohlen, Interaktionen mit der Datenbank mittels Transaktionen zu verarbeiten. Falls auf Transaktionen verzichtet wird, haben wir beobachtet, dass neue Datensätze direkt in die Datenbank geschrieben werden, sobald die Objekte persistiert wurden. Gemäß der DataNucleus-Dokumentation werden Datensätze aktualisiert, sobald der Persistenzmanager geschlossen wird.

Datenbankabfragen können mit verschiedenen Abfragesprachen durchgeführt werden; der Entwickler hat die Wahl zwischen SQL, JDOQL, JPOXSQL und JPQL. Das Werkzeug kann außerdem in der JDO-Implementierung mit Vererbungsstrukturen in den Datenmodellen umgehen und bietet dem Entwickler einige Vererbungsstrategien an, die festlegen, auf welche Art und Weise die Vererbung in der relationalen Datenbank abgebildet werden soll. Nennenswert ist ebenfalls, dass DataNucleus nicht nur relationale Datenbanken, sondern mit entsprechenden Plug-Ins auch nicht-relationale Datenbanken, unterstützt. Beispielsweise können Datensätze in objektbasierten Datenbanken wie db4o oder auch in XML-Dokumenten persistiert werden.
Die größte Schwäche von DataNucleus ist die unzureichende INSERT-Performanz bei der Verarbeitung der maximalen Last des Mengengerüsts. In den von uns durchgeführten

Messungen auf dem Car-PC waren 70% der Speichervorgänge nicht in der vorgesehenen Zeit geschehen, weswegen nicht alle erforderlichen Operationen auf der Datenbank in der Gesamtzeit ausgeführt werden konnten (siehe Listing 2). Leider hatten auch einige Performanz-Tweaks (z.B. kein L2-Caching) keinen spürbaren Effekt und brachten zum Teil eine noch schlechtere Performanz als der Testlauf ohne jegliche Optimierungen.

```
|--------------------|-------|---------|-------|---------|-------------|
| Measurement Point  |   #   | Average |  Min  |   Max   |    Total    |
|--------------------|-------|---------|-------|---------|-------------|
| Datapoint Insert   | 95200 |   6.174 | 3.952 | 236.894 | 587,766.950 |
|--------------------|-------|---------|-------|---------|-------------|
| Position Insert    |   476 |   6.469 | 4.234 |  21.685 |   3,079.440 |
|--------------------|-------|---------|-------|---------|-------------|
```

Listing 2: Ergebnis des Testlaufs (vgl. Kriterium Performanz der INSERT-Abfragen)

Durch die vielen Funktionen und Eigenarten von DataNucleus wie beispielsweise die Einführung eines Enhancers bedarf es vergleichsweise viel Zeit bis das Werkzeug lauffähig ist und der Entwickler sich eingearbeitet hat. Auch die Konformität zum JDO-Standard wirkt sich negativ auf die Einarbeitungszeit aus.

Die Query-Syntax ist darüber hinaus nicht einheitlich. So werden INSERT- und UPDATE-Befehle über Funktionen realisiert; für SELECT-Abfragen hingegen muss sich der Entwickler mit einer Datenbanksprache wie JDOQL befassen. Das Einfügen und Bearbeiten von Datensätzen werden über denselben Befehl durchgeführt, wobei beim Bearbeiten der Datensatz zunächst vom Entity-Manager über den Primärschlüssel angefordert werden muss.

Falls der Java-Code von der Datenbasis abweicht, muss sich der Entwickler selbst um das Anpassen der Datenbank kümmern. DataNucleus bietet keine speziellen Mechanismen für die Migration des Schemas oder die Daten an. Ferner ist es mit DataNucleus nicht möglich, die Antwort bei Datenbankfehlern festzulegen. Sollte ein unerwarteter Fehler auftreten, wird dieser dem Entwickler stets über Exceptions mitgeteilt.

Kurz gesagt bietet DataNucleus zwar zahlreiche Features für Entwickler an, die man bei anderen Kandidaten vermisst, kommt aber nicht an deren Performanz heran.

PRO	**CONTRA**
⊕ Großer Funktionsumfang	⊖ Schlechte INSERT-Performanz
⊕ Unterstützung von Persistenzstandards	⊖ Hoher Einarbeitungsaufwand

6.3.3 ORMLite

Untersuchte Version

4.25 (22.08.11)

Lizenz

Eigene OS-Lizenz

Webseite

ormlite.com

Bei ORMLite [ORM11] wurde vom einzigen Entwickler bereits während der Entwicklung großen Wert auf einen leichtgewichtigen OR-Mapper gelegt. Das Resultat ist ein Werkzeug, das außer dem eigentlichen Kern nur eine zusätzliche Bibliothek zur Unterstützung der Datenbank durch einen JDBC-Treiber benötigt. Komplexe Features wie die Migration von existierenden Schemas sind nicht implementiert und es wurde von Anfang an darauf geachtet, dass kein großer Ballast durch Komplexität, wie man ihn beispielsweise bei Hibernate vorfindet, anfällt. Dennoch bietet ORMLite trotz des kleinen Kerns eine Reihe von nützlichen Funktionen.

ORMLite richtet sich nach dem Entwurfsmuster Data Access Object (DAO) [Bou11], das durch die Kapselung von Datenbankzugriffen das Austauschen der zugrundeliegenden Datenbank ermöglicht, ohne dabei den aufrufenden Java-Code dabei verändern zu müssen. Die eigentliche Programmlogik wird hierbei strikt von den technischen Details der Datenspeicherung getrennt; die Datenbankoperationen werden isoliert von der Programmlogik ausgeführt, wodurch diese flexibel einsetzbar ist. Jedes DAO bietet in ORMLite Methoden zum Anlegen, Löschen und Bearbeiten von Datensätzen an und wickelt in einem einzigen Objekt alle Datenbankinteraktionen pro Klasse ab (siehe Listing 3 für ein Code-Beispiel). Der Entwickler verwaltet daher die Datenspeicherung bei mehreren Modellen auch mit unterschiedlichen Entity-Managern.

```
String cs = "jdbc:postgresql://localhost:5432/ORMLite";
JdbcConnectionSource jdbc = new JdbcConnectionSource(cs,"user","pw");
Dao<Device,Long> dao = DaoManager.createDao(jdbc,Device.class);
dao.create(new Device());
```

Listing 3: Speichern eines *Device*-Objekts mit Hilfe eines DAOs

Die Dokumentation von ORMLite macht auf den ersten Blick einen unzureichenden Eindruck. So ist das „Getting-Started“-Dokument zu kurz und man vermisst wichtige Details; sucht man aber konkrete Stichworte, zeigt sich schnell, dass alle relevanten Aspekte ausreichend und auf verständliche Weise beschrieben werden. Ein große Community über das Werkzeug lässt sich nicht ausmachen: Es ist eine Mailing-Liste vorhanden, auf der gelegentlich Fragen auch innerhalb weniger Tage beantwortet werden. Im Schnitt muss man jedoch länger warten um eine Antwort zu erhalten. Auch ein umfangreicher Support kann nicht erwartet werden, da es sich bei dem Autor der Software im Gegensatz zu manchen Konkurrenzwerkzeugen um eine einzelne Person handelt. Selbstverständlich steht es jedem frei, den Autor des OR-Mappers direkt zu kontaktieren.

Die Abbildung der Datenmodelle auf die relationale Datenbank erfolgt standardmäßig mit Hilfe von Java-Annotationen, wie es beispielsweise in Listing 4 für das *Device*-

Modell gemacht wurde. Alternativ lässt sich das Mapping der Java-Klassen auch direkt im Quellode vollziehen. Wer das Spring-Framework verwendet, findet auf den Seiten von ORMLite zusätzlich einige Beispiele, die zeigen, wie die Mapping-Konfiguration anhand der XML-Dateien des Spring-Frameworks erfolgen kann.

```
@DatabaseTable
class Device {

  // primary key
  @DatabaseField(generatedId = true)
  private long id;

  @DatabaseField
  private String urn;

  @DatabaseField
  private String name;

  @DatabaseField
  private String description;

  @DatabaseField
  private Type type;

  /* ... */
}
```

Listing 4: OR-Mappings des *Device*-Modells mit Java-Annotationen

Weiterhin steht dem Entwickler eine so genannte „Query-Builder“-Klasse zur Verfügung, mit dem SELECT-Abfragen ohne eine Abfragesprache auf der Datenbank ausgeführt werden können. Dabei geht die Flexibilität von gewöhnlichen SELECT-Abfragen im SQL-Format nicht verloren. Das Anlegen und Bearbeiten von Datensätzen eines Modells geschieht dabei über verschiedene Methoden eines Data Access Objects. Darüber hinaus können „rohe“ SQL-Abfragen direkt auf der Datenbank ausgeführt werden, falls der Funktionsumfang des Query-Builders nicht ausreichen sollte.

Es gibt bei ORMLite ferner eine Reihe von Werkzeugen, die bei der Erstellung des Schemas und der Tabellen helfen. Zum einen kann mit Hilfe der *createTable*-Methode aus der *TableUtils*-Klasse die benötigten Tabellen automatisch generiert werden, zum anderen kann durch den Aufruf einer anderen Methode derselben Klasse das SQL-Schema auf der Konsole ausgegeben werden. Ein weiterer Pluspunkt ist, dass das Werkzeug einfache Vererbungsstrukturen im Datenmodell auf eine relationale Datenbank abbilden kann, indem das Werkzeug alle Basisklassen prüft und die gefundenen Attribute in die Tabelle der Unterklasse aufnimmt.

Außerdem erleichtert ORMLite die Java-Entwicklung auf Android-Betriebssystemen, in-

dem es eine spezielle Bibliothek anbietet, die direkt auf die Android-Datenbank-API zugreift und so als OR-Mapper auf mobilen Endgeräten zum Einsatz kommen kann. Diese spezielle Bibliothek ist dabei unabhängig von der JDBC-Schnittstelle zu benutzen, da standardmäßig auf Android-Endgeräten keine JDBC-Treiber vorhanden sind und diese speziell auf die Android-Runtime angepasst werden müssen.

Bei der Entwicklung der Anwendung muss auf manche Besonderheiten des OR-Mapping-Werkzeugs ORMLite geachtet werden. Beispielsweise ist zu berücksichtigen, dass vom Werkzeug stets ein Konstruktor ohne Parameter für jedes definierte Datenmodell gefordert ist. Außerdem gab es beim Testen des Werkzeugs mit einer PostgreSQL-Datenbank Probleme mit dem *BigDecimal*-Typ, der erst gesondert behandelt werden musste, bevor eine persistente Speicherung des gesamten Modells möglich war. Des Weiteren muss beachtet werden, dass nur Attribute mit nicht primitiven Typen als Fremdschlüssel deklariert werden können. Die automatische Vergabe einer ID erfolgt bei ORMLite über die Option *generatedId* der *DatabaseField*-Annotation.
Wer zudem Abweichungen zwischen Modell und Datenbank hat, muss diese selbst in der Datenbank beheben, da ORMLite keine Unterstützung bei der Migration des Schemas und der Daten anbietet. Bei Abweichungen und bei Fehlern im Allgemeinen werden stets Exceptions geworfen. Ein Rückgabeverhalten im Fehlerfall lässt sich mit der ORMLite-API folglich nicht einstellen.

Insgesamt betrachtet macht ORMLite einen stabilen Eindruck und ist durch die Unterstützung zahlreicher DBMS vielfältig einsetzbar. Der Funktionsumfang hingegen kann die Ansprüche des Industriepartners nicht völlig zufriedenstellen. Weitere Funktionalitäten lassen sich bei ORMLite nur direkt im Quellcode einbauen. Das speziell auf Leichtgewichtigkeit ausgelegte Werkzeug kann im Gegensatz zu DataNucleus ohne Probleme mit der maximalen Last des Mengengerüsts umgehen.

PRO	**CONTRA**
⊕ Fokus auf Leichtgewichtigkeit	⊖ Geringer Funktionsumfang
⊕ Schnelle Inbetriebnahme	⊖ Kleine Community und nur ein Entwickler

6.3.4 Persist

Persist [Per11] wurde von Beginn an als minimalistischer und schlanker OR-Mapper konzipiert. Der Code umfasst deshalb insgesamt nur 12 Klassen und verwendet oftmals Java-Standardklassen anstatt eigene Methoden bereitzustellen. Dadurch sollen eine hohe Performanz, einfache Benutzbarkeit und leichte Integrierbarkeit erreicht werden. Ein weiterer Vorteil davon ist, dass keine weiteren Abhängigkeiten entstehen. Außer der Kernbibliothek wird nur der Datenbanktreiber benötigt, um Persist in Betrieb zu nehmen.

Untersuchte Version
1.1.1 (03.02.11)

Lizenz
BSD

Webseite
github.com/rufiao/persist

Die Dokumentation von Persist beschränkt sich auf eine einzige README-Datei, die nur noch durch die selbst zu erstellende JavaDoc-API-Dokumentation erweitert wird. Durch den minimalistischen Ansatz ist jedoch nicht viel mehr nötig, um die Funktionalität zu verstehen. Innerhalb weniger Minuten hat man die erste persistente Klasse erstellt und die Daten in der Datenbank gespeichert.
Außer dem in der Web-Entwicklungsplattform GitHub[1] vorhandenen Issue-Tracker beschränkt sich die Community auf eine kaum verwendete Mailing-Liste und selbst dort scheint der Support durch den einzigen Entwickler auf Grund langer Antwortzeiten und wenig Aktivität durch andere Mitglieder eher eingeschränkt zu sein.

Die Integration von Persist ist denkbar einfach. Die Verbindung zur Datenbank wird über die Java-Standardklasse *Connection* aufgebaut, die nach erfolgreicher Verbindung nur noch an Persist übergeben werden muss. Danach kann man mit einem einzelnen Funktionsaufruf die Daten einer Klasse in der Datenbank speichern. Voraussetzung dafür ist jedoch, dass man das Schema der Datenbank der Klasse entsprechend bereits manuell angelegt hat, da Persist den Benutzer in dieser Richtung nicht unterstützt. Besitzt die Klasse keine automatisch aufsteigenden IDs, so kommt Persist ganz ohne Annotationen oder XML-Konfiguration aus. Sollte eine solche ID benötigt werden, so kann mit Hilfe einer einzigen Annotation diese Funktion hinzugefügt werden. Dadurch ist der Übergang von einer Anwendung ohne Datenbankpersistenz hin zum Speichern der Daten in der Datenbank schnell und simpel zu bewältigen.

[1] http://www.github.com

Eine kompakte Beispielanwendung könnte zum Beispiel folgendermaßen aussehen:

```
class Device {

  private long id;
  private String urn;
  private String name;
  private String description;
  private Type type;

  @Column(autoGenerated=true)
  public long getId() {
    return id;
  }

  public static void main(String[] args) {
    Device dev;
    /* Device erstellen und Daten einfügen */

    try {
      Class.forName("org.postgresql.Driver");
    } catch (java.lang.ClassNotFoundException e) { }

    Connection connection;
    try {
      connection = DriverManager.getConnection(
        "jdbc:postgresql://localhost:5432/persist", "postgres", "orm");
    } catch (java.sql.SQLException e) {}

    Persist persist = new Persist(connection);
    persist.save(dev);
  }

  /* ... */
}
```

Listing 5: Beispielanwendung mit Persist

Um Datenbankabfragen durchzuführen, verwendet Persist Standard-SQL, gibt jedoch auf Wunsch nicht nur komplette Klassen zurück, sondern kann die erhaltenen Daten auch automatisch in HashMap-Listen einsetzen. Auch einzelne Werte können direkt abgefragt werden. Zudem beschränkt sich die Verwendung von SQL nicht auf Abfragen. Es dürfen auch andere SQL-Befehle direkt über Persist an den Server gesendet werden.

Als eines von wenigen fortgeschrittenen Features unterstützt Persist immerhin Transaktionen. Dazu muss das Werkzeug allerdings erst in einen speziellen Modus versetzt werden, in dem man automatische Commits deaktiviert. Dadurch wirkt dieses Feature im Vergleich mit anderen Werkzeugen etwas unintuitiv.

Viele weitere Features wie zum Beispiel die Unterstützung bei der Schemagenerierung oder die Unterstützung von Vererbungsstrukturen fielen der Minimalität zum Opfer. Dadurch kann Persist nur über seine Einfachheit und Performanz mit den anderen Werkzeugen konkurrieren.

PRO	**CONTRA**
⊕ Einfache Integration	⊖ Wenige Features
⊕ Hohe Performanz	⊖ Keine Community und nur ein Entwickler

6.3.5 Siena

Untersuchte Version
1.0.0-b5 (24.06.11)

Lizenz
Apache 2.0

Webseite
sienaproject.com

Siena [Sie11] wurde nicht nur für die Verwendung mit SQL-Datenbanken ausgelegt, sondern soll mit demselben Code auch No-SQL-Datenbanken unterstützen. Momentan wird dabei unter anderem die Google App Engine unterstützt. Es wird zur Zeit jedoch daran gearbeitet, auch andere Datenbanken zu unterstützen. Um die Möglichkeiten, die diese Datenbanken bieten, ausnutzen zu können, ist es jedoch zum Teil notwendig, kleinere Anpassungen vorzunehmen bzw. ab und zu direkt auf den Datenbanktreiber zuzugreifen.

Ein weiteres Alleinstellungsmerkmal ist die Möglichkeit, Daten sowohl über einen Persistenzmanager in der Datenbank zu speichern als auch das Active-Record-Pattern zu verwenden. Bei letzterem muss die Klasse, die die Daten enthält, von der von Siena bereitgestellten *Model*-Klasse abgeleitet werden. Die Verwendung des Persistenzmanagers erhöht den Aufwand jedoch ein wenig, da für jede persistente Klasse eine eigene Instanz des Persistenzmanagers verwendet werden muss und man deshalb bei einer hohen Anzahl von Klassen schnell den Überblick verliert oder sich noch Behelfsfunktionen schreiben muss. Die *Model*-Klasse bietet zudem noch einige Komfortfunktionen wie z.B. bereits implementierte *equals*- und *hashCode*-Funktionen, die die persistenten Felder nutzen, um z.B. zwei Instanzen einer Klasse zu vergleichen.
In beiden Betriebsarten bietet Siena einige Annotationen an, um Relationen zwischen verschiedenen Tabellen darzustellen, unter anderem Many-to-many, One-to-many und One-to-one. Auch Vererbungsstrukturen stellen kein Problem dar. Für jede Subklasse und die Oberklasse benutzt Siena dabei eine einzelne Tabelle.

Die von Siena bereitgestellte *Query*-Klasse bietet Methoden, um Datenbankabfragen zusammenzustellen. Dabei wird durch eine Verkettung von Funktionsaufrufen die Syntax von SQL nachgestellt. Dies erlaubt auch komplexe Abfragen, ohne Kenntnisse von SQL beim Entwickler vorauszusetzen .
Um zum Beispiel ein bestimmtes *Device*-Objekt aus der Datenbank zu bekommen, könnte der Code folgendermaßen aussehen:

```
Query<Device> q = Device.all(Device.class);
Device dev = q.search("device key", "uin").get();
```

Listing 6: Beispielquery mit Siena

Eine rudimentäre Unterstützung bietet Siena bei der Generierung des Schemas. Dazu werden jedoch zum einen die Apache DdlUtils[2] und deren Abhängigkeiten benötigt, zum anderen ist einiges an Handarbeit notwendig, um das Schema zu erstellen. Darüber

[2] http://db.apache.org/ddlutils/

hinausgehende Funktionen müssten direkt über die DdlUtils programmiert werden.

Zur Verbindungsaufnahme mit der Datenbank nutzt Siena eine Properties-Datei, die alle Einstellungen wie Treiber, Host, Username und Passwort enthält. Diese Datei muss Teil der JAR-Datei sein, was die Konfigurierung zur Laufzeit erschwert. Dafür muss innerhalb des Programmcodes keinerlei Konfiguration durchgeführt werden. Wird zudem noch das Active-Record-Pattern verwendet, so kann sofort mit dem Abspeichern oder Holen von Daten begonnen werden, ohne extra eine Verbindung mit der Datenbank aufbauen zu müssen.

Die Community von Siena nutzt unter anderem eine Mailing-Liste und von GitHub zur Verfügung gestellte Tools wie den Issue-Tracker und das Wiki. Auf der Mailing-Liste ist nicht nur der Hauptentwickler aktiv, sondern auch weitere Entwickler und Nutzer. Auf Fragen erhält man schnell eine Antwort.
Über das Wiki lässt sich am einfachsten auf die Dokumentation von Siena zugreifen. Diese ist momentan jedoch in einer Umgestaltungsphase und enthält deshalb noch nicht alle Informationen. Eine per JavaDoc generierte API-Dokumentation existiert nicht. Würde man sie selbst generieren, so wäre sie jedoch auch nicht vollständig, da viele Funktionen über keinerlei oder nur wenige Kommentare verfügen.

In unseren Tests stellte sich das vom Entwickler angegebene Feature, dass keine weiteren Abhängigkeiten eingebunden werden müssten, als falsch heraus. Ohne eine Bibliothek aus dem Fundus von Apache Commons gaben unsere Laufversuche Fehler über nicht gefundene Klassen aus. Nach deren Einbindung war das weitere Einrichten von Siena einfach zu bewerkstelligen, eine solche falsche Information ist für Einsteiger jedoch sehr irritierend.

PRO	**CONTRA**
⊕ Sowohl Entity-Manager als auch Active-Record	⊖ Externe Bibliothek nötig
⊕ Einfache Abfragen	⊖ Schemagenerierung unnötig kompliziert

6.4 Resultat

Die vorherige Beschreibung der OR-Mapper spiegelt das Resultat der Evaluationsphase wieder. Dieses kann in kompakter Form der nachfolgenden Tabelle entnommen werden.

Kriterium	*Cayenne*	*DataNucleus*	*ORMLite*	*Persist*	*Siena*
Performanz der INSERT-Abfragen	⊕	⊖	⊕	⊕	⊕
Dokumentation/Support	⊕	⊕	⊕	⊙	⊙
Simplizität des Abspeicherns	⊕	⊕	⊙	⊕	⊙
Generierung des Datenbankschemas	⊕	⊕	⊕	⊖	⊕
Anlegen von Datenbanken	⊕	⊕	⊕	⊖	⊕
Einarbeitungsaufwand	⊙	⊖	⊕	⊖	⊙
Form der Modelldefinition	⊖	⊕	⊕	⊕	⊕
Community	⊙	⊕	⊙	⊖	⊙
Vielfältiger Einsatz	⊙	⊕	⊙	⊙	⊙
Syntax von Abfragen	⊕	⊖	⊕	⊖	⊕
Umgang mit INSERT- und UPDATE-Abfragen	⊕	⊕	⊖	⊖	⊕
Vererbungsstrukturen zwischen Datenmodellen	⊕	⊕	⊕	⊖	⊕
Transaktionen	⊕	⊕	⊕	⊕	⊕
Antwort bei Fehlschlägen	⊖	⊖	⊖	⊕	⊖
Entity-Manager	⊖	⊕	⊕	⊕	⊕
Unterstützung bei Schemamigration	⊕	⊖	⊖	⊖	⊕
Unterstützung bei Datenmigration	⊖	⊖	⊖	⊖	⊖
Gesamtpunktzahl	94	90	95	53	**96**
Endnote	2	2-	2	5	**2**

Tabelle 4: Resultat der Evaluation auf einen Blick

7 Empfehlung

Das Endergebnis unserer Untersuchung ist sehr eng. Mit drei Werkzeugen, die jeweils nur einen Punkt auseinander- und damit alle auf derselben Note liegen und einem weiteren Werkzeug, das nur auf Grund seiner mangelhaften Performanz ausschied, ist das Teilnehmerfeld sehr eng beieinander. Einziger Ausreißer ist der OR-Mapper Persist, der aufgrund seines geringen Funktionsumfangs viele Anforderungen nicht erfüllen konnte.

Für das Nutzungsszenario der Cinovo AG wird von uns **Siena** als Werkzeug empfohlen. Es ist zum einen das Werkzeug mit der höchsten Punktzahl, zum anderen das mit der geringsten Anzahl an nicht erfüllten Kriterien.

Durch die Möglichkeit, sowohl Persistenzmanager als auch das Active-Record-Pattern zu verwenden, bietet es eine größere Flexibilität als die anderen untersuchten Lösungen. Neue Modelle sind leicht ohne die Hilfe eines GUI-Editors zu erstellen. Ein solcher wird beispielsweise bei Apache Cayenne vorausgesetzt. Durch die Unterstützung von NoSQL-Datenbanken ist Siena auch zukunftssicher, da solche Datenbanken in immer mehr Bereichen eingesetzt werden. Außer Siena bietet nur DataNucleus über Erweiterungen eine solche Unterstützung an. Im Gegensatz zu diesem Werkzeug ist Siena jedoch eines der schnellsten was die für das Szenario relevante Performanz von INSERT-Abfragen angeht.

Eine der wenigen Schwachstellen stellt die Dokumentation dar, die sich momentan in einer Überarbeitungsphase befindet und ständig verbessert wird. Hier konnten die anderen Werkzeuge besser punkten. Durch die einfache Syntax und gute „Getting Started“-Dokumente wird dieser Punkt jedoch großteils ausgeglichen und Lücken in der Dokumentation finden sich hauptsächlich bei komplexeren Themen wie Relations.

Siena bot insgesamt das rundeste Angebot der fünf untersuchten Werkzeuge und ist deshalb unsere Empfehlung.

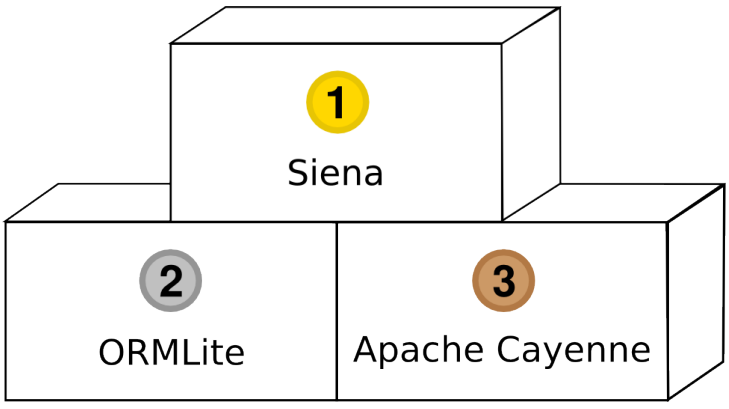


Abbildung 4: Die Siegertreppe unserer Fachstudie

A Versionshistorie

Version 1.0 (24.11.2011)

- ▷ Erarbeitung der Empfehlung (7)
- ▷ Syntaktische und stilistische Korrektur des gesamten Dokuments

Version 0.4 (29.10.2011)

- ▷ Steckbriefe zu allen Werkzeugen erstellt (6.3)
- ▷ Ermittlung der Endnoten aller begutachteten Werkzeuge (6.4)
- ▷ Verbesserungen an der Beschreibung des Bewertungsschemas (6.2)
- ▷ Erstellung eines Abkürzungsverzeichnisses (B)

Version 0.3 (21.09.2011)

- ▷ Beschreibung des Nutzungsszenarios verbessert (4)
- ▷ Mengengerüst des Industriepartners aufgenommen (4.1)
- ▷ Überarbeitung der Bewertungskriterien (5)
- ▷ Grundstruktur des Kapitels Evaluation festgelegt (6)
- ▷ Beschreibungstext zur Voruntersuchung erstellt (6.1)
- ▷ Erklärungen zur Shortlist-Matrix aufgenommen (6.1)
- ▷ Bewertungsschema erstellt (6.2)
- ▷ Vorlagen für die Steckbriefe erstellt (6.3)
- ▷ Erste Resultate der Evaluation aufgenommen (6.4)

Version 0.2 (20.08.2011)

- ▷ Vervollständigen der Titelseite und Korrektur der Seitennummerierung
- ▷ Neues Layout des Marktüberblicks inklusive Logos (3)
- ▷ Überarbeitung des Nutzungsszenarios (4)
- ▷ Exaktere Definitionen und Ergänzungen der Bewertungskriterien (5)
- ▷ Aufnahme von irrelevanten Bewertungskriterien (5.3)
- ▷ Erstellung der Shortlist auf Basis der K.O.-Kriterien (6.1)

Version 0.1.1 (30.07.2011)

- ▷ Ganzheitliche Korrekturen an der Dokumentstruktur
- ▷ Grobe Skizzierung der Nutzungsszenarios (4)
- ▷ Einbinden des Kriterienkatalogs (5)
- ▷ Ausformulieren von relevanten Bewertungskriterien und K.O.-Kriterien (5)

Version 0.1 (30.06.2011)

- ▷ Erstellen der ersten Fassung der Ausarbeitung
- ▷ Einbinden eines tabellarischen Marktüberblicks über die vorhandenen OR-Mapping-Werkzeuge (3)

B Abkürzungsverzeichnis

API	Application Programming Interface
DAO	Data Access Object
DBMS	Datenbankmanagementsystem
ERM	Entity-Relationship-Modell
GUI	Graphical User Interface
JAR	Java Archive
J2ME	Java Platform Micro Edition
JDBC	Java Database Connectivity
JDK	Java Development Kit
JDO	Java Data Objects
JPA	Java Persistence API
OR	Object-Relational
ORM	Object-Relational Mapping
POJO	Plain Old Java Object
Shortlist	Engere Auswahl von Werkzeugen für die Evaluationsphase
SoPra	Softwarepraktikum
SQL	Structured Query Language
XML	Exstensible Markup Language

C Abbildungsverzeichnis

D Tabellenverzeichnis

E Quellenverzeichnis

[Amb03] Ambler, Scott W.: *Agile Database Techniques*. John Wiley & Sons, 2003

[BGP11] Buchgraber, Christian ; Gildein, Philipp ; Pirrung, Philipp: *aidGer - Hilfskraftmittelverwaltungssystem*. http://www.aidger.de. Version: 2011

[Bou11] Boughton, Alex: Object Relational Database Mapping, Computer Science, University of Colorado at Boulder, Vereinigte Staaten von Amerika, 2011

[Cay11] *Apache Cayenne*. http://cayenne.apache.org. Version: 13. Oktober 2011

[Cin11] *Cinovo AG*. http://www.cinovo.de. Version: 22. November 2011

[Dat11] *DataNucleus*. http://www.datanucleus.org. Version: 23. September 2011

[ORM11] *ORMLite*. http://www.ormlite.com. Version: 28. Oktober 2011

[Ors06] Orsag, Jaroslav: *Object-Relational Mapping*, Comenius University, Bratislava, Slovakia, Diplomarbeit, 2006

[Per11] *Persist*. https://github.com/rufiao/persist. Version: 13. September 2011

[PT11] PostgreSQL-Team: *PostgreSQL - DBMS*. http://www.postgresql.org. Version: 26. August 2011

[Rus08] Russell, Craig: Bridging the object-relational divide. In: *ACM Queue* 6 (2008), 07, Nr. 3, S. 18–28

[Sch10] Scheit, Philipp: *Analyse und Lösungen für den Object-relational Impedance Mismatch*, Goethe-Universität, Frankfurt am Main, Diplomarbeit, 2010

[Sie11] *Siena*. http://www.sienaproject.com. Version: 10. November 2011

Erklärung

Hiermit versichern wir, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Christian Buchgraber, Philipp Gildein, Philipp Pirrung)