INFORMATIONSTECHNOLOGIE UND ÖKONOMIE
Herausgegeben von Wolfgang Gaul, Armin Heinzl und Martin Schader 32

Matthias Merz

Konzeptioneller Entwurf und prototypische Implementierung einer Sicherheitsarchitektur für die Java Data Objects-Spezifikation

ANWENDUNGEN
PROBLEME
WISSEN

PETER LANG
Internationaler Verlag der Wissenschaften

Vorwort

Die vorliegende Dissertationsschrift ist während meiner Zeit als Doktorand am Lehrstuhl für Wirtschaftsinformatik III an der Universität Mannheim entstanden. Für die Unterstützung meiner fachlichen Interessen gilt mein besonderer Dank meinem Doktorvater Herrn Prof. Dr. Martin Schader, der mein Promotionsvorhaben stets unterstützt und das Fortschreiten meiner Arbeit im konstruktiven Dialog gefördert hat. Bei Herrn Prof. Dr. Daniel Veit bedanke ich mich für die Übernahme des Korreferats. Ferner danke ich auch den weiteren Prüfern Herrn Prof. Dr. Colin Atkinson und Herrn Prof. Dr. Carsten Trenkler.

Bei Herrn Dr. Heinz Kredel möchte ich mich für die langjährige Unterstützung und die hilfreiche Diskussion meiner Ideen am Rechenzentrum der Universität Mannheim bedanken. Mein Dank gilt auch dem Leiter der Einrichtung, Herrn Dr. Hans Günther Kruse, der die organisatorischen Rahmenbedingungen im Rechenzentrum geschaffen hat, die zum Gelingen der Promotion erheblich beigetragen haben. Zudem danke ich allen Kollegen des Lehrstuhls für Wirtschaftsinformatik III, die mich durch zahlreiche Anregungen während des Doktorandenseminars unterstützt und meine Dissertation im Rahmen gemeinsamer Projekte vorangetrieben haben. Vor allem möchte ich mich bei Herrn Dr. Markus Aleksy bedanken, der immer wieder die Zeit für wertvolle fachliche Gespräche fand und stets ein kompetenter Ansprechpartner war. Ferner danke ich Herrn Dr. Ivica Dus sowie Clarissa Dold, Wolfgang Fiedler, Matthias Gutheil, Martina Hey, Manfred Sabo und Daniela Winklmeier, die auf unterschiedliche Weise zum Erfolg meiner Promotion beigetragen haben.

Abschließend möchte ich mich herzlich bei meinen Eltern Hildegard und Reinhold Merz sowie meiner Schwester Melanie bedanken, die mich während meiner Studien- und Promotionszeit immer motiviert und in vielfältiger Weise unterstützt haben.

Mannheim im August 2007 Matthias Merz

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ANSI	American National Standards Institute
API	Application Programming Interface
CLI	Call Level Interface
CRUD	Create, Retrieve, Update, Delete
DAC	Discretionary Access Control
DBMS	Datenbankmanagementsystem
DFG	Default Fetch Group
DSD	Dynamic Separation of Duty
EC	Executive Committee
EJB	Enterprise Java Beans
FCO	First Class Object
IT	Informationstechnologie
J2EE	Java 2 Enterprise Edition
JAAS	Java Authentication and Authorization Service
JACC	Java Authorization Contract for Containers
JAXP	Java API for XML Processing
Java EE	Java Enterprise Edition
Java SE	Java Standard Edition
JCA	Java Cryptography Architecture
JCP	Java Community Process
JDBC	Java Database Connectivity
JDK	Java Development Kit
JDO	Java Data Objects
JDO-ID	JDO-Identität
JDOQL	Java Data Objects Query Language
JNI	Java Native Interface
JPA	Java Persistence API
JSR	Java Specification Request
JSSE	Java Secure Socket Extension
JVM	Java Virtual Machine
MAC	Mandatory Access Control
NIST	National Institute of Standards and Technology
OIF	Object Interchange Format
ODL	Object Definition Language
ODMG	Object Data Management Group
OMG	Object Management Group
OODBMS	Objektorientiertes Datenbankmanagementsystem
OQL	Object Query Language

PAM	Pluggable Authentication Modules
RBAC	Role-Based Access Control
RDBMS	Relationales Datenbankmanagementsystem
RI	Referenzimplementierung
SCO	Second Class Object
SHA	Secure Hash Algorithm
SoC	Separation of Concerns
SPI	Service Provider Interface
SQL	Structured Query Language
SQLJ	Structured Query Language for Java
SSD	Static Separation of Duty
SSL	Secure Sockets Layer
RMI	Remote Method Invocation
TCK	Technology Compatibility Kit
TLS	Transport Layer Security
UML	Unified Modeling Language
URL	Uniform Resource Locator

1 Einführung

Die Bedeutung moderner Informations- und Kommunikationssysteme (IuK-Systeme) zur Unterstützung betrieblicher Funktionen und Prozesse nimmt seit Jahren beständig zu. Standen zunächst die effiziente Erfassung, Übermittlung, Speicherung und Aufbereitung von Informationen als zentrale Aufgaben der IuK-Systeme im Vordergrund, zählte zwischenzeitlich die Nutzung von Web-Technologien zur Unterstützung des *Electronic Business* zu den wesentlichen Herausforderungen im Unternehmenskontext. Angesichts einer gestiegenen Sicherheitssensibilisierung werden aktuell auch vermehrt Maßnahmen zur Durchsetzung von Sicherheits- und Datenschutzzielen beim Einsatz von IuK-Systemen diskutiert (vgl. Seiler 2006). Aufgrund der zunehmenden Verbreitung heterogener Datenbank- und Speichertechnologien und der Verwendung zahlreicher Schnittstellen, Standards und Anfragesprachen rücken ferner die Themengebiete *Enterprise Application Integration* und Datenintegration, beispielsweise im Umfeld der betrieblichen Managementunterstützung (*Business Intelligence*), immer häufiger in den Mittelpunkt der Betrachtung (vgl. z. B. Weimer 2006).

Um einen einfachen, einheitlichen und transparenten Zugriff auf beliebige Datenbank- und Speichertechnologien zu ermöglichen, wurde für die Programmiersprache Java der Industriestandard Java Data Objects (JDO) verabschiedet (vgl. Java Community Process 2004, Java Community Process 2006h). Die JDO-Spezifikation definiert eine datenbankunabhängige Programmierschnittstelle sowie eine einheitliche Anfragesprache und gewährleistet u. a. die Austauschbarkeit verschiedener Datenbankmanagementsysteme. Mit dem vorrangigen Einsatz der JDO-Technologie lässt sich die Anzahl unterschiedlicher Datenbankschnittstellen und Anfragesprachen deutlich reduzieren. Aufgrund einer konsequenten Ausrichtung auf die Thematik Persistenz bleiben sicherheitsspezifische Fragestellungen in der JDO-Spezifikation weitgehend unberücksichtigt. Die vorliegende Dissertationsschrift greift diesen Sachverhalt auf und verdeutlicht, dass das Fehlen technischer Sicherheitsmechanismen auf der Ebene der JDO-Implementierung ein erhebliches Sicherheitsrisiko darstellt. Aufbauend auf dieser Erkenntnis verfolgt diese Schrift die primäre Zielsetzung, mit Hilfe eines konzeptionellen Entwurfs und der prototypischen Implementierung einer Sicherheitsarchitektur für die Java Data Objects-Spezifikation die bestehenden Sicherheitsdefizite wirksam zu eliminieren.

Im nachfolgenden Abschnitt wird zunächst die Bedeutung der Informationssicherheit im Unternehmenskontext thematisiert. Anschließend werden die konkrete Zielsetzung sowie die Abgrenzung gegenüber weiteren Problembereichen behandelt und die Gliederung der weiteren Kapitel erläutert.

1.1 Motivation

Der umfassende Einsatz von rechnergestützten Informations- und Kommunikationssystemen in allen betrieblichen Funktionsbereichen hat gegenwärtig zu einer hohen Abhängigkeit der Unternehmen von der Verfügbarkeit ihrer Systeme geführt. Überdies zählt auch die Gewährleistung von Vertraulichkeit, Integrität und Verbindlichkeit im Rahmen eines effektiven *Security Managements* immer häufiger zu den wesentlichen Determinanten des Unternehmenserfolgs. Der Länder- und Unternehmensgrenzen überschreitende Zugriff auf bestehende Systeme und Daten z. B. über das Angebot von Web-Services für Lieferanten, den Betrieb von Internet-Portalen für Kunden, sowie das Bereitstellen von Mobile Business-Lösungen für Außendienstmitarbeiter hat jedoch bis heute nicht zu einer ausreichenden Sensibilisierung der Unternehmen gegenüber möglichen Sicherheitsproblemen beigetragen. Eine weltweit angelegte Studie von Ernst & Young belegt in diesem Zusammenhang, dass sich die Diskrepanz zwischen den gewachsenen Sicherheitsrisiken einerseits und den Bemühungen der Unternehmen, Informationssicherheit auf ausreichendem Niveau zu gewährleisten, in den letzen Jahren weiter verstärkt hat (Ernst & Young 2005). Erst in jüngster Zeit nimmt die Informationssicherheit in den Unternehmen einen sukzessiv höheren Stellenwert ein (vgl. Seiler 2006).

Durch das zielgerichtete Ausnutzen vorhandener Sicherheitslücken konnten in der Vergangenheit oftmals unternehmensinterne Daten unrechtmäßig von Dritten eingesehen, manipuliert oder gelöscht werden. Beispielsweise gab die US-Aktiengesellschaft Retail Ventures Inc. im April 2005 bekannt, dass „Unbekannte“ die Kreditkartendaten von mehr als 1,4 Millionen Kunden der Einzelhandelskette DSW Shoe Warehouse gestohlen hatten (Heise Online 2005). Im August 2006 teilte das Telekommunikationsunternehmen AT&T mit, dass Angreifern der Einbruch in ein internes Computersystem gelungen war und ihnen so der Zugriff auf persönliche Informationen und Kreditkartendaten von ca. 19.000 Kunden möglich war (Heise Online 2006b). Im Dezember 2006 meldete auch die Universität von Kalifornien, dass ein Computerhacker in eine Datenbank eingedrungen war und Zugang zu persönlichen Informationen wie Namen, Adressen oder Sozialversicherungsnummern von nahezu 800.000 Studenten, Dozenten und Beschäftigten hatte (Heise Online 2006a).

Die aufgeführten Fälle machen deutlich, dass ein unberechtigter Zugriff auf unternehmensinterne Datenbanken für eine Unternehmung, Mitarbeiter oder Kunden erhebliche Folgen haben kann. Neben einer nachhaltigen Schädigung der Reputation können die wirtschaftlichen Auswirkungen u. U. existenzbedrohend sein. Gelingt etwa der Diebstahl von Forschungsergebnissen, Geschäftsgeheimnissen oder Kundendaten, sind die wirtschaftlichen Folgen unabsehbar. Erfolgt z. B. eine Weitergabe vertraulicher Daten an Dritte, können Schadensersatzforderungen und strafrechtliche Konsequenzen die Folge sein. Unterstellt man infolge eines Angriffs auch die teilweise Löschung von Datenbeständen, muss neben einer zeitaufwändigen Rekonstruktion der Daten in der Regel auch von einer Betriebsablaufstörung und damit verbundenen Umsatzausfällen ausgegangen werden. Angesichts dieser Risiken und aufgrund rechtlicher Verpflichtungen gegenüber Mitarbeitern, Gesellschaftern oder Dritten im Umgang mit Unternehmensdaten sollte Informationssicherheit grundsätzlich als Aufgabe der Unternehmensführung aufgefasst werden (Göbel 2005).

Effektive Informationssicherheit kann jedoch nicht ausschließlich durch organisatorische Planung, Umsetzung und Kontrolle erreicht werden. Hoppe und Prieß differenzieren in diesem Zusammenhang zwischen technischen und nicht-technischen Sicherheitsmaßnahmen

(2003, S. 61–268). Im Bereich nicht-technischer Maßnahmen verweist Schlienger beispielsweise auf eine stärkere Berücksichtigung des menschlichen Verhaltens und fordert höhere Investitionen in die Sensibilisierung und Ausbildung von Mitarbeitern (2006). Bezogen auf technische Aspekte entwickelten Saltzer und Schroeder bereits vor mehr als 30 Jahren verschiedene Prinzipien zur Konstruktion sicherer Systeme (1975), deren Kernaussagen bis heute Gültigkeit besitzen (Eckert 2005). Hierzu zählen u. a. die Konzepte *Separation of Duty* (Aufgabenteilung, siehe Abschnitt 2.2.2) und *Principle of Least Privilege* (Zuteilung minimaler Berechtigungen). Aufgrund der stetigen Zunahme neuer Technologien und deren kontinuierlicher Weiterentwicklung sind jedoch nur beschränkt allgemein verbindliche Empfehlungen zur Vermeidung technisch bedingter Sicherheitsrisiken möglich. Weitergehende Maßnahmen lassen sich häufig nur in Abhängigkeit konkret verwendeter Produkte und Standards, wie z. B. Programmiersprachen, Kommunikationsprotokolle, Betriebssysteme, Verschlüsselungsalgorithmen und Datenbankschnittstellen, ergreifen. Erschwerend kommt hinzu, dass spezielle Methodiken zur Konstruktion sicherer IT-Systeme (Informationstechnologie Systeme) „[...] im Sinne eines systematischen Security Engineerings [...]“ bislang kaum entwickelt wurden (Eckert 2005, S. 67).

Betriebliche Anwendungssysteme setzen in der Regel eine dauerhafte Datenhaltung voraus. In der Programmiersprache Java stehen einem Entwickler zur Realisierung dieser Aufgabe zahlreiche Ansätze zur Verfügung (siehe Abschnitt 3.4). Mit Verabschiedung der Java Data Objects-Spezifikation im Jahr 2002 wurde ein Industriestandard für Objektpersistenz etabliert, der Anwendungsentwicklern erstmals einen einfachen, einheitlichen und transparenten Zugriff auf persistente Objekte gewährleistet. Ein wesentliches Kennzeichen dieser Technologie ist, dass JDO-basierte Anwendungen ohne Quellcode-Modifikationen mit beliebigen Datenbank- und Speichertechnologien betrieben werden können. Beispielsweise gestattet der Einsatz von JDO die wahlweise Nutzung eines objektorientierten oder relationalen Datenbankmanagementsystems ebenso wie eine direkte Verwaltung persistenter Objekte in einem Dateisystem. Aufgrund weiterer technischer Eigenschaften, wie z. B. der Bereitstellung eines Object-Caches und einer Transaktionsverwaltung auf Objekt-Ebene, sowie der Verfügbarkeit zahlreicher Implementierungen, stellt JDO einen geeigneten Ansatz zur Realisierung von Objektpersistenz in Java dar.

Wird die JDO-Technologie hinsichtlich potentieller Sicherheitsdefizite analysiert, ergibt sich dagegen eine abweichende Beurteilung. Aufgrund der Konzeption als leichtgewichtiger Persistenzansatz schreibt die JDO-Spezifikation keine Integration technischer Sicherheitsmaßnahmen auf der Ebene einer JDO-Implementierung vor. Hierdurch bedingt, lassen sich nach Aufbau einer Datenbankverbindung beliebige Objekte ohne weitere Berechtigungsprüfung rekonstruieren, modifizieren oder löschen (siehe Abschnitt 4.4). Vor dem Hintergrund der oben skizzierten Bedeutung der Informationssicherheit kann der Einsatz der JDO-Technologie im Unternehmenskontext daher nicht ohne das Ergreifen technischer Sicherheitsmaßnahmen befürwortet werden.

Eine JDO ergänzende Sicherheitsarchitektur, welche den Zugriff auf persistente Objekte kapselt und auf Basis eines benutzer- und rollenbasierten Berechtigungskonzepts anwendungsübergreifend Zugriffsbeschränkungen etabliert, würde die mit JDO verbundenen Vorteile in betrieblichen Anwendungen nutzbar machen und zugleich die aufgezeigten Sicherheitsrisiken eliminieren. Aufbauend auf dieser Erkenntnis wird im nachfolgenden Abschnitt die Zielsetzung dieser Dissertationsschrift formuliert und gegenüber weiteren Problembereichen abgegrenzt.

1.2 Zielsetzung und Abgrenzung

Diese Schrift verfolgt das Ziel, die konzeptuell bedingten Sicherheitsrisiken der Java Data Objects-Spezifikation durch Bereitstellung einer geeigneten Sicherheitsarchitektur zu eliminieren und die JDO-Technologie unter Gewährleistung von Informationssicherheit auch im Kontext betrieblicher Anwendungen nutzbar zu machen. Den Ausgangspunkt bilden hierbei eine mögliche Unternehmensentscheidung für den Einsatz von JDO und die sich daran anknüpfende Fragestellung, wie die konzeptionell bedingten Sicherheitsdefizite der JDO-Architektur eliminiert werden können.

Als Lösungsansatz wird eine Sicherheitsarchitektur vorgeschlagen, welche die Vergabe und Prüfung individueller Berechtigungen für Benutzer bzw. deren Rollen gestattet. Diese sieht vor, dass die Rechte, mit deren Hilfe sich der Zugriff auf persistente Objekte über die Schnittstellen der JDO-Spezifikation regeln lässt, anwendungsübergreifend gültig sind.

Eine wesentliche Voraussetzung zur Akzeptanz der skizzierten Sicherheitsarchitektur stellt ihre Unabhängigkeit gegenüber einer konkret verwendeten JDO-Implementierung dar. Lösungsansätze, die lediglich Implementierungen einzelner Hersteller um diverse Sicherheitsfunktionen erweitern, würden die Portabilität JDO-basierter Anwendungen übermäßig einschränken. Zudem müssen Änderungen an den Schnittstellen der JDO-Spezifikation weitgehend vermieden werden, da andernfalls zeit- und kostenintensive Anpassungen notwendig werden. Zur Gewährleistung von Informationssicherheit muss letztlich auch sichergestellt sein, dass eine direkte Interaktion zwischen JDO-basierter Anwendung und JDO-Implementierung verhindert und somit eine „Umgehung“ der skizzierten Sicherheitsmechanismen ausgeschlossen werden kann.

Eine Sicherheitsarchitektur, die diesen Bedingungen genügt, kann dazu beitragen, ein wesentliches Hindernis für den Einsatz der JDO-Technologie im betrieblichen Kontext abzubauen.

1.3 Gliederung

Die vorliegende Dissertationsschrift umfasst sieben Kapitel. Im Verlauf des ersten Kapitels wurden bereits die Motivation sowie die Zielsetzung und Abgrenzung erläutert.

Im zweiten Kapitel werden die sicherheitstechnischen Grundlagen und Konzepte besprochen, die maßgeblich in die Konzeption und prototypische Implementierung der Sicherheitsarchitektur für die Java Data Objects-Spezifikation eingeflossen sind. Es werden die technischen Sicherheitsmaßnahmen Authentifizierung und Autorisierung, das Role-Based Access Control-Pattern sowie das Pluggable Authentication Modules-Framework besprochen. Anschließend erfolgt die Betrachtung der Java-Sicherheitsarchitektur und des Java Authentication and Authorization Service.

Das dritte Kapitel führt in die Grundlagen der Objektpersistenz ein und stellt Persistenzmodelle und deren Charakterisierungsmerkmale sowie das Konzept der transparenten Persistenz vor. Bei der anschließenden Betrachtung unterschiedlicher Ansätze zur Realisierung von Objektpersistenz in Java zeigt sich, dass diese in der Regel lediglich den Einsatz

bestimmter Datenbankmanagementsysteme gestatten und jeweils separate Schnittstellen und Anfragesprachen nutzen.

Kapitel vier stellt die Java Data Objects-Spezifikation vor und unterstreicht, dass Anwendungsentwicklern mit dem Einsatz der JDO-Technologie der transparente Umgang mit persistenten Objekten ermöglicht wird. Es wird dargelegt, dass sich JDO gegenüber anderen Persistenzlösungen insbesondere durch die Unterstützung beliebiger Datenbank- und Speichertechnologien auszeichnet. Nach einer kurzen Einordnung der Entwicklung von JDO im Rahmen des Java Community Process erfolgt die Vorstellung der JDO-Architektur. Hierbei werden das Persistenzmodell, der Lebenszyklus von JDO-Instanzen sowie die JDO-Schnittstellen erläutert. Anschließend werden die Sicherheitsdefizite der JDO-Architektur herausgearbeitet, gefolgt von einer kritischen Würdigung der JDO-Spezifikation.

Im fünften Kapitel werden der konzeptuelle Entwurf sowie die prototypisch implementierte Sicherheitsarchitektur für die JDO-Spezifikation vorgestellt. Nach einer Darlegung der primären Zielsetzung und der weiteren Designziele erfolgt im Kontext einer konzeptionellen Betrachtung die Herleitung eines grundsätzlichen Lösungsansatzes. Es wird aufgezeigt, dass sich die skizzierten Sicherheitslücken der JDO-Architektur durch Integration eines geeigneten Authentifizierungs- und Autorisierungsverfahrens wirksam schließen lassen. Die softwaretechnische Umsetzung des konzeptionellen Entwurfs wird nachfolgend betrachtet. Hierbei werden u. a. die Authentifizierungs- und Autorisierungskomponenten sowie das Managementsystem zur Verwaltung der Benutzer-, Rollen- und Berechtigungsdaten vorgestellt. Ein Abschnitt, in dem die wichtigsten Erkenntnisse zusammengefasst werden, vervollständigt das Kapitel.

Das sechste Kapitel beschreibt die Evaluation der Sicherheitsarchitektur. Anhand unterschiedlicher Kriterien wird die prototypisch implementierte Sicherheitsarchitektur hinsichtlich ihrer Praxistauglichkeit bewertet. Neben der Beurteilung des Integrationsaufwands und des Laufzeitverhaltens JDO-basierter Anwendungen wird mit Hilfe einer typischen Beispielanwendung aus dem *Electronic Commerce*-Umfeld untersucht, inwieweit mit dem Einsatz der prototypischen Implementierung die vorhandenen Sicherheitslücken JDO-basierter Anwendungen wirksam geschlossen werden können. Das Kapitel schließt mit der Vorstellung des Evaluationsergebnisses.

Kapitel sieben fasst die wesentlichen Ergebnisse dieser Dissertationsschrift zusammen und gibt einen Ausblick auf zukünftige Entwicklungsmöglichkeiten.

3 Grundlagen der Objektpersistenz

Wurden im vorherigen Kapitel die sicherheitstechnischen Grundlagen besprochen, so werden in diesem Kapitel für ein umfassendes Verständnis der Java Data Objects-Spezifikation sowie der im Rahmen dieser Dissertation entstandenen Sicherheitsarchitektur die Grundlagen der Objektpersistenz vertieft.

Objektorientierte Programmiersprachen gestatten die Konstruktion komplex strukturierter Objekte, deren Existenz im Allgemeinen an den Lebenszyklus einer Ablaufumgebung gebunden ist. Werden Objekte zur Laufzeit einer Anwendung erzeugt und spätestens mit Terminierung der Ablaufumgebung zerstört, so nennt man diese Instanzen *transiente* Objekte. Objekte, die dieser Beschränkung nicht unterliegen und bis zu ihrer expliziten Löschung etwa in einem objektorientierten Datenbankmanagementsystem (OODBMS) die Terminierung ihrer Ablaufumgebung überdauern, werden in der Literatur als *persistente Objekte* bezeichnet (vgl. Atkinson 1991, S. 455 sowie Schmidt 1991, S. 24). Sind persistente Objekte in einem Sekundärspeicher „ausgelagert“, lassen sich diese zu einem späteren Zeitpunkt *(zeitliche Persistenz)* oder außerhalb des aktuellen Adressraums *(räumliche Persistenz)* rekonstruieren. In der Programmiersprache Java können Objekte eine Terminierung der JVM überdauern, wenn diese zuvor beispielsweise in einer Datei serialisiert werden (siehe Abschnitt 3.4.1).

Nach Ansicht von Saake, Schmitt und Türker darf der Begriff „Persistenz“ allerdings nicht mit dem Abspeichern von Objekten unter der Kontrolle eines Anwendungsprogramms gleichgesetzt werden. Vielmehr müsse nach Auffassung der Autoren die dauerhafte und anwendungsübergreifende Verwaltung der Objekte im Mittelpunkt stehen, wie dies z. B. bei Nutzung eines OODBMS der Fall ist (Saake, Schmitt und Türker 2003, S. 162). Allerdings müssen hierzu Funktionen zum Erzeugen, zum Ändern und zum Löschen persistenter Objekte sowie für den Zugriff auf diese zur Verfügung stehen. Die vier elementaren Zugriffsfunktionen werden auch als CRUD-Operationen bezeichnet, wobei sich das Akronym CRUD aus den Wörtern *Create*, *Retrieve*, *Update* und *Delete* zusammensetzt.

In den folgenden Abschnitten wird zunächst auf die Persistenz im Kontext objektorientierter Programmiersprachen eingegangen und anschließend Persistenzmodelle sowie deren Charakterisierungsmerkmale vorgestellt. Weiter wird das Konzept der transparenten Persistenz erläutert und abschließend Ansätze zur Realisierung von Objektpersistenz in der Programmiersprache Java besprochen.

3.1 Integration von Programmiersprache und Datenbank- bzw. Speichertechnologie

Eine notwendige Voraussetzung zur Realisierung von Objektpersistenz ist die geeignete Integration der Programmiersprache und der Datenbank- bzw. Speichertechnologie. In der

Literatur werden diesbezüglich verschiedene Lösungsansätze erörtert. Eine mögliche Vorgehensweise stellt die Verwendung einer prozeduralen Programmierschnittstelle dar, die im englischen Sprachgebrauch auch als *Call Level Interface* (CLI) bezeichnet wird. Ein CLI definiert Funktionen für den Zugriff auf ein Datenbankmanagementsystem (DBMS), die innerhalb einer Programmiersprache aufgerufen werden können. Beispielsweise repräsentiert das Java Database Connectivity-API (JDBC, vgl. Java Community Process 2006g) eine solche Programmierschnittstelle, über die in Java der Zugriff auf zahlreiche relationale Datenbankmanagementsysteme (RDBMS) ermöglicht wird. Einen weiteren Ansatz stellt die „Einbettung" spezieller Datenbank-Befehle in eine Programmiersprache dar. Der SQLJ-Standard (American National Standard for Information Technology 2003) gestattet z. B. die direkte Verwendung von SQL-Anweisungen im Java-Quellcode, die allerdings mit Hilfe eines Präprozessors vor dem Übersetzungslauf in korrekte Java-Syntax transformiert werden müssen (vgl. z. B. Saake und Sattler 2003, S. 145-194).

Aufgrund der „Kluft" zwischen den jeweils in der Programmiersprache und dem Datenbanksystem verwendeten Typsystemen beanstanden Saake und Sattler die Verwendung der skizzierten Ansätze zur Realisierung von Objektpersistenz. Für eine „tatsächliche Integration" von Programmiersprache und Datenbanksystem sehen sie lediglich zwei Alternativen (2003, S. 309). Eine Möglichkeit stellt demnach die Erweiterung des Typsystems einer Programmiersprache um ein Datenbankmodell und die zugehörigen Datenbankanweisungen dar. In diesem Fall sprechen die Autoren von Datenbankprogrammiersprachen, welche sie weiter in relationale und objektorientierte Datenbankprogrammiersprachen unterteilen. In Zusammenhang mit der Nutzung relationaler Datenbanken skizzieren Saake et al. beispielhaft die mögliche Integration des Datentyps `Relation` in eine Programmiersprache und die Bereitstellung entsprechender Zugriffsmethoden auf die korrespondierenden Datenbankrelationen (2003, S. 175). Die zweite Herangehensweise sieht die Verwendung des Typmodells einer Programmiersprache als Datenbankmodell vor. Dieser Ansatz wird in der deutschsprachigen Literatur auch unter dem Begriff *persistente Programmiersprachen* erörtert (vgl. Saake et al. 2003, S. 175 sowie Saake und Sattler 2003, S. 309), wogegen im Englischen die Bezeichnung *Persistent Application Systems* gebräuchlich ist (Atkinson und Morrison 1995, S. 319).

Bereits 1983 definierten Atkinson, Bailey, Chisholm, Cockshott und Morrison spezielle Anforderungen, deren Einhaltung die Autoren in Zusammenhang mit der Entwicklung persistenter Programmiersprachen forderten. Im Jahr 1995 überarbeiteten Atkinson und Morrison diese Anforderungen und entwickelten hieraus das Paradigma der *orthogonalen Persistenz* (siehe Abschnitt 3.2). Dieses Paradigma bildete die Grundlage zur Entwicklung der persistenten Programmiersprache PJava (Atkinson, Jordan, Daynès und Spence 1996), die später unter der Bezeichnung PJama weiter entwickelt wurde (Atkinson und Jordan 2000a, Atkinson 2001). Für eine vollständige Umsetzung des Paradigmas der orthogonalen Persistenz in Java unterbreiteten Atkinson und Jordan überdies auch Vorschläge zur Novellierung der Java-Spezifikation sowie zur partiellen Anpassung zentraler Java-Bibliotheken (2000b).

3.2 Persistenzmodelle und deren Charakterisierungsmerkmale

Zur Realisierung von Objektpersistenz bedarf es der geeigneten Umsetzung entsprechender Konzepte, die häufig unter dem Begriff „Persistenzmodell“ zusammengefasst werden. Nach Reese bestimmt ein Persistenzmodell, wie Komponenten in einer Datenbank abgebildet werden (2003, S. 20). Munro ergänzt diese Definition: „The principal tenet of the persistence model is that it abstracts over all the physical properties of data such as how long it is stored, where it is stored, how it is stored, what form it is kept in and who is using it“ (1993).

Persistenzmodelle lassen sich anhand unterschiedlicher Fragestellungen charakterisieren. Hierzu zählen beispielsweise, ob Objekte unabhängig von ihrem Typ persistenzfähig sind, wie sich der Wechsel eines Objekts in einen persistenten Zustand auf eventuell vorhandene, über Referenzen erreichbare transiente Instanzen auswirkt und inwiefern der Quellcode einer Anwendung gleichermaßen für den Umgang mit persistenten und transienten Objekten geeignet ist. Die drei aufgeführten Fragestellungen gehen auf die Arbeit von Atkinson und Morrison zurück (1995). Anhand dieser Fragestellungen lassen sich typabhängige und typunabhängige (siehe Abschnitt 3.2.1), implizite und explizite (siehe Abschnitt 3.2.2) sowie abhängige und unabhängige Persistenzmodelle (siehe Abschnitt 3.2.3) unterscheiden. Die Autoren forcieren entsprechend die Nutzung typunabhängiger Persistenzmodelle, die sowohl eine implizite Persistenzpropagierung gestatten, als auch die Eigenschaft der persistenten Unabhängigkeit aufweisen. Wie in Abbildung 8 dargestellt, lassen sich diese drei Kriterien zum Paradigma der orthogonalen Persistenz zusammenführen. Die Bezeichnung „orthogonal“ ergibt sich hierbei aus der angestrebten Unabhängigkeit der Persistenzeigenschaft gegenüber sämtlichen weiteren Eigenschaften und Konzepten der verwendeten Programmiersprache. Die Verletzung eines der drei genannten Kriterien führt auch zum Verlust der Orthogonalität (Atkinson und Morrison 1995, S. 330). Als zentralen Vorteil orthogonaler Persistenz wird in der Literatur genannt, dass sich ein Anwendungsentwickler ohne Berücksichtigung von Persistenzaspekten vollständig auf das Design und die Implementierung der Anwendungslogik konzentrieren kann (vgl. Atkinson und Jordan 2000a). Saake und Sattler sehen in der orthogonalen Persistenz jedoch auch einige Nachteile, zu denen sie u. a. die fehlende Möglichkeit zur Generierung optimierter Datenbankanfragen zählen (2003, S. 317).

Die genannten Kriterien zur Klassifizierung von Persistenzmodellen werden im Folgenden detailliert betrachtet. In der Literatur diskutiert man auch weitere Charakterisierungsmerkmale von Persistenzmodellen (vgl. z. B. Saake et al. 2003), auf die in Abschnitt 3.2.4 ebenfalls eingegangen wird.

3.2.1 Typabhängige vs. typunabhängige Persistenz

Eine Möglichkeit zur Klassifikation von Persistenzmodellen besteht in der Betrachtung der Persistenzfähigkeit von Objekten in Abhängigkeit ihres Typs. Typabhängige Persistenz bezeichnet demnach ein Persistenzmodell, welches zwischen persistenzfähigen und nicht persistenzfähigen Typen unterscheidet (Atkinson und Morrison 1995, S. 329–330).

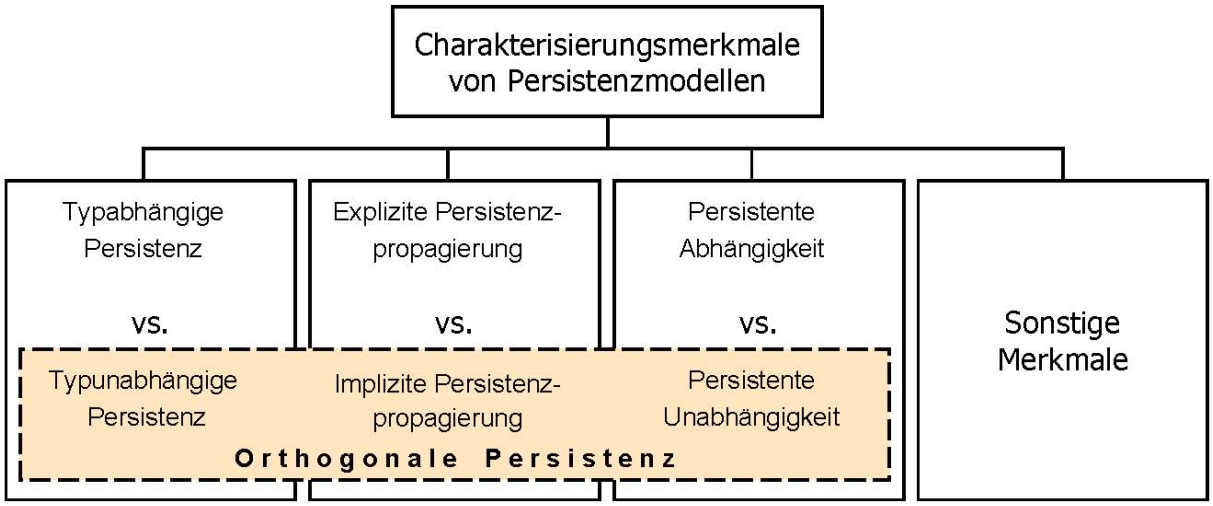

Abbildung 8: Persistenzmodelle und deren Charakterisierungsmerkmale

In einem Persistenzmodell, das dieses Konzept umsetzt, sind lediglich Objekte ausgewählter Klassen und gegebenenfalls primitiver Typen persistenzfähig. Heuer unterscheidet zur Realisierung typabhängiger Persistenz die Ansätze „Persistenzfähigkeit durch Vererbung“ und „Persistenzfähigkeit durch explizite Kennzeichnung“ (1997, S. 524–525). Im Fall der „Persistenzfähigkeit durch Vererbung“ erbt eine Klasse die Persistenzeigenschaft von einer speziellen Basisklasse. In Programmiersprachen, die, wie z. B. Java, keine Mehrfachvererbung unterstützen, schränkt diese Vorgehensweise die Entwicklung persistenzfähiger Anwendungen stark ein. Im Fall der „Persistenzfähigkeit durch explizite Kennzeichnung“ ist typabhängige Persistenz auch über die Nutzung eines bestimmten Schlüsselworts zur Markierung persistenzfähiger Klassen realisierbar. Sofern dieses Schlüsselwort nicht bereits in der Programmiersprache selbst definiert ist, kann beispielsweise ein zusätzlicher Präprozessor die persistent markierte Klasse um entsprechende Funktionalität erweitern. Sieht ein Persistenzmodell lediglich typabhängige Persistenz vor, sind Objekte bestimmter Typen nicht persistenzfähig. Sofern erforderlich, muss ein Entwickler für diese Typen eine zusätzliche Persistenzlösung implementieren, was jedoch nicht immer möglich ist und zudem dem *Separation of Concerns*-Paradigma (SoC) widerspricht (vgl. Lopes und Hursch 1995).

Typunabhängige Persistenz bezeichnet demgegenüber ein Persistenzmodell, das sich durch die Persistenzfähigkeit aller Objekte unabhängig von deren Typ auszeichnet. Dieser Ansatz wird in der Literatur häufig auch als *typorthogonale* Persistenz bezeichnet. Bei gegebener Typ-Orthogonalität ist ein Anwendungsentwickler nicht auf die Verwendung bestimmter persistenzfähiger Klassen beschränkt. Sind sämtliche Objekte einer Programmiersprache persistenzfähig, kann dies je nach Implementierung jedoch auch negative Auswirkungen zeigen. Werden beispielsweise die Instanzen mehrerer Klassen ausschließlich als transiente Objekte verwendet, kann der Persistenz-*Overhead* ohne einen konkreten Nutzen die Effizienz der Speicherverwaltung beeinträchtigen (vgl. Saake et al. 2003, S. 164–165).

3.2.2 Implizite vs. explizite Persistenzpropagierung

Bei der impliziten bzw. expliziten Persistenzpropagierung wird die Fragestellung betrachtet, wie sich der Übergang eines Objekts in einen persistenten Zustand auf weitere, über Objektreferenzen abhängige Instanzen auswirkt (vgl. Atkinson und Morrison 1995, S. 330). Für transitiv abhängige Objekte, die sich in transientem Zustand befinden, muss festgelegt sein, ob diese mit dem Zustandswechsel des Ausgangsobjekts automatisch in den Sekundärspeicher abgebildet werden.

Bei der expliziten Persistenzpropagierung ändert sich der Status der transitiv erreichbaren Objekte zunächst nicht. Transitiv abhängige Objekte müssen, sofern erforderlich, über einen expliziten Vorgang in diesen Zustand überführt werden. Wird hierauf verzichtet, kann die Rekonstruktion des Ausgangsobjekts zu einer Ausnahmesituationen führen, wenn über dessen Objektreferenzen auf eine aktuell nicht im Speicher befindliche Instanz navigiert wird. Sollen zudem sämtliche transitiv abhängigen Objekte später bzw. außerhalb des aktuellen Adressraums zugreifbar sein, muss folglich die Anwendung die Übertragung dieser Objekte in den Sekundärspeicher explizit anstoßen. Ist der Objektgraph hierbei in einer zyklischen Struktur angeordnet, ist zudem sicherzustellen, dass Objekte nicht mehrfach in den Sekundärspeicher ausgelagert werden. Andernfalls ist die semantisch korrekte Wiederherstellung des Objektgraphs kaum möglich.

Bei der impliziten Persistenzpropagierung wechselt hingegen jedes Objekt, das sich innerhalb der transitiven Hülle eines persistenten Objekts befindet, automatisch in einen persistenten Zustand. Daher wird dieser Ansatz in der Literatur häufig als transitive Persistenz bzw. im englischen Sprachgebrauch auch als „persistence by reachability“ bezeichnet. Von der transitiven Persistenz sind solche Objekte ausgenommen, die explizit durch den Anwendungsentwickler als transient markiert werden. In einem Persistenzmodell, welches typabhängige Persistenz (vgl. Abschnitt 3.2.1) und implizite Persistenzpropagierung miteinander kombiniert, lassen sich zudem lediglich die Objekte persistenzfähiger Typen automatisch in einen Sekundärspeicher abbilden. In der Programmiersprache Java werden beide Konzepte beispielsweise beim Ansatz der Objektserialisierung entsprechend umgesetzt (vgl. Abschnitt 3.4.1).

3.2.3 Persistente Abhängigkeit vs. persistente Unabhängigkeit

Das Prinzip der persistenten Unabhängigkeit bezeichnet die Eigenschaft, den Umgang mit transienten und persistenten Objekten in Anwendungen ohne jegliche Differenzierung zu ermöglichen. Der zugrundeliegende Quellcode einer Anwendung ist folglich in beiden Fällen identisch (Saake und Sattler 2003, S. 310–311). Atkinson hebt in diesem Zusammenhang hervor, dass sich die Semantik einer Programmiersprache, beispielsweise hinsichtlich der Auswertungsreihenfolge, der Typsicherheit, des Zugriffsschutzes oder der Objektidentität, nicht ändern darf (2001). Saake und Sattler verweisen daher darauf, dass im Fall persistenter Unabhängigkeit einem Anwendungsentwickler keine expliziten „store“- oder „move“-Kommandos zum expliziten Anstoßen von Speicher- oder Ladevorgängen zur Verfügung stehen dürfen (2003, S. 310). In der Literatur wird persistente Unabhängigkeit gelegentlich auch am Beispiel einer Sortierfunktion verdeutlicht, die als Übergabeparameter eine Menge von Objekten erwartet (Atkinson und Morrison 1995, S. 329). Bei

Einhaltung der persistenten Unabhängigkeit können der Sortierfunktion sowohl transiente als auch persistente Instanzen übergeben werden.

3.2.4 Weiterführende Charakterisierungsmerkmale

Neben Typ-Orthogonalität, Persistenzpropagierung und persistenter Unabhängigkeit werden weitere Vorschläge zur Charakterisierung von Persistenzmodellen diskutiert. Nach Saake et al. lassen sich Persistenzmodelle beispielsweise auch hinsichtlich einer statischen oder dynamischen Persistenzierung unterscheiden (2003, S. 165-166). Bei der statischen Persistenzierung werden Objekte automatisch als persistente Objekte erzeugt, wohingegen im Rahmen der dynamischen Persistenzierung Objekte zu einem beliebigen Zeitpunkt in ihrem Lebenszyklus zwischen den Zuständen transient und persistent wechseln können. Ebenso fassen Saake et al. die Bereitstellung von Klassenextensionen als ein weiteres Unterscheidungskriterium von Persistenzmodellen auf, über die sich verhältnismäßig einfach alle persistenten Objekte einer Klasse verwalten lassen. Die Autoren betonen in diesem Zusammenhang die herausgehobene Bedeutung persistenter Klassenextension im Kontext objektorientierter Anfragesprachen (2003, S. 165).

Mit der Umsetzung eines Persistenzmodells in unterschiedlichen Softwareprodukten lassen sich für diese weitere Unterscheidungskriterien formulieren. Jordan führt beispielsweise die Eigenschaft der Wiederverwendbarkeit des Programmcodes an (2004, S. 5), „[...] since any mechanism-specific modifications to code, whether directly or indirectly, may likely render that code unusable in another context“. Diesbezüglich verweist der Autor auf Persistenzansätze, die beispielsweise das Einbinden spezieller Pakete erfordern und somit einer angestrebten Portabilität der Anwendung („write once, run anywhere“, vgl. Curtin 1998) entgegenstehen. Weitere Charakterisierungsmerkmale sieht Jordan zudem in Bezug auf die Performance, die Skalierbarkeit insbesondere im Umfeld eingebetteter Systeme (*Embedded Systems*) sowie die Unterstützung von Transaktionen (2004, S. 5–6). Auch die Unterstützung mehrerer Klassenversionen sowie den Aufwand zum Erstellen und Ausführen persistenter Anwendungen (*Operational Complexity*) nennt Jordan als mögliche Kriterien. Zur Beurteilung des zuletzt genannten Kriteriums fordert Jordan die Berücksichtigung von „[...] resources such as file systems, raw disks, operating system processes, and any additional complexity added to the environment [...] such as special classpath settings or run-time properties“ (2004, S. 7). Bauer und King nennen zudem die Hersteller-Abhängigkeit sowie die Unabhängigkeit der Persistenzlösung gegenüber speziellen Datenbank- und Speichertechnologien als mögliche Kriterien zur Charakterisierung von Persistenzlösungen (2006, S. 30).

3.3 Transparente Persistenz

Die softwaretechnische Umsetzung der Konzepte Typ-Orthogonalität, implizite Persistenzpropagierung und persistente Unabhängigkeit hat sich in der Vergangenheit häufig als schwer realisierbar erwiesen. Saake und Sattler verweisen auf die Studien von Atkinson und Morrison in Zusammenhang mit der Programmiersprache Java und nennen diesbezüglich verschiedene Beispiele (2003, S. 311-312). Neben datenbankspezifischen Problemen,

wie z. B. der Implementierung einer geeigneten Pufferverwaltung, zeigen die Autoren u. a. auf, dass das Konzept der persistenten Unabhängigkeit für die Programmiersprache Java nicht vollständig zu realisieren ist. Auch die Umsetzung der Typ-Orthogonalität erweist sich in Java als komplexe Aufgabe, da hierzu der Einsatz einer modifizierten JVM notwendig wird (vgl. Jordan 2003). Entsprechende Bemühungen haben Marquez, Blackburn, Mercer und Zigman unternommen, deren prototypische Umsetzung jedoch erhebliche Performance-Probleme aufweist (2001). In der Praxis hat sich daher das leichter zu realisierende Konzept der transparenten Persistenz durchgesetzt, das als abgeschwächte Form der orthogonalen Persistenz aufgefasst werden kann. In der Literatur ist der Begriff der transparenten Persistenz uneinheitlich definiert. Kappel und Schröder setzen transparente Persistenz beispielsweise mit dem Konzept der persistenten Unabhängigkeit (vgl. Abschnitt 3.2.3) gleich (1998, S. 411–412). Bei Korb hingegen steht der vereinfachte Zugriff auf persistente Objekte im Mittelpunkt: „The idea behind transparent persistence is that it simplifies the programming model involved in storing and retrieving Java objects in a back-end data store“ (2004). Eine hilfreiche Definition gelingt Roos in Zusammenhang mit der Beschreibung des JDO-Standards (2002, S. 54–55):

> „The term transparent persistence refers to:
>
> 1. The illusion that all persistent instances are in memory and immediately available.
> 2. The implicit update of dirty persistent instances with the data store upon transaction commit.
> 3. The automatic mapping of Java types to the native types of the underlying data store.“

Bauer und King verweisen im Zusammenhang mit der transparenten Persistenz auf die Bedeutung des SoC-Paradigmas: „We use transparent to mean a complete separation of concerns between the persistent classes of the domain model and the persistence logic, where the persistent classes are unaware of – and have no dependency on – the persistence mechanism“ (2006, S. 112). Dies schließt folglich nicht unmittelbar den Einsatz spezieller Code-Fragmente oder Metadaten zur Realisierung von Objektpersistenz aus, sofern sich durch deren Verwendung keine Auswirkungen auf das Domänen-Modell ergeben.

3.4 Ansätze zur Realisierung von Objektpersistenz in Java

Die Umsetzung eines Persistenzmodells in eine konkrete Implementierung erweist sich im Allgemeinen als komplexe Aufgabe. Zahlreiche Anforderungen, wie z. B. Zuverlässigkeit, Transaktionsverwaltung, Mehrbenutzerzugriff, Zugriffsschutz und Performance sind hierbei zu berücksichtigen. Weitere Schwierigkeiten ergeben sich auch hinsichtlich der Unterstützung verschiedener Datenbank- und Speichertechnologien, dem unterschiedlichen Antwortzeitverhalten zwischen Primär- und Sekundärspeicher sowie der Umsetzung transparenter Persistenz (vgl. Abschnitt 3.3). Die folgenden Unterabschnitte gehen auf

ausgewählte Ansätze zur Realisierung von Objektpersistenz in der Programmiersprache Java ein. Da diese Auswahl keinen Anspruch auf Vollständigkeit erhebt, sei als Ergänzung auf Jordan (2004) verwiesen.

3.4.1 Objekt-Serialisierung

Die Objektserialisierung stellt für die Programmiersprache Java einen einfachen und generell verfügbaren Mechanismus zur Realisierung von Objektpersistenz dar (Sun Microsystems 2003). Über den expliziten Methodenaufruf `writeObject(Object)` einer `ObjectOutputStream`-Instanz lassen sich Objekte, die das `java.io.Serializable`-Interface implementieren, in einen systemunabhängigen Bytestrom umwandeln (serialisieren). Eine manuelle Implementierung der `writeObject(Object)`-Methode durch den Anwendungsentwickler ist hierbei nicht erforderlich. Java sieht allerdings die Erweiterung serialisierbarer Klassen vor und ermöglicht die Bereitstellung klassenspezifischer `writeObject(ObjectOutputStream)` bzw. `readObject(ObjectOutputStream)` Methoden. Der generierte Bytestrom lässt sich beispielsweise in eine Datei umleiten oder zur Realisierung räumlicher Persistenz mit Hilfe eines Sockets innerhalb eines Netzwerks übertragen. Gemäß dem Paradigma der transitiven Persistenz (vgl. Abschnitt 3.2.2) erfolgt die Berücksichtigung des vollständigen Objektgraphs und somit die Serialisierung sämtlicher direkt und indirekt erreichbarer Objekte. Hierbei werden für jedes Objekt der Objekttyp, die Identität sowie die Zustände aller nicht explizit als transient markierten Instanzvariablen berücksichtigt. Der in Java implementierte Mechanismus verhindert zudem, im Fall einer zyklischen Struktur des Objektgraphs, die mehrfache Serialisierung einzelner Objekte (Sun Microsystems 2003, S. 2–3). Bei der Deserialisierung wird der Bytestrom über die Methode `readObject()` zum Lesen geöffnet, und die serialisierten Objekte werden innerhalb der JVM rekonstruiert. Hierzu wird zunächst ein dem Datentyp entsprechendes Objekt erzeugt und anschließend werden die Instanzvariablen rekonstruiert und Objektreferenzen gesetzt. Eine einfache Versionskontrolle stellt hierbei sicher, dass die aktuell vorhandene Klasse mit dem deserialisierten Objekt kompatibel ist.

Mit der Verfügbarkeit von Java in der Version 1.4 wird standardmäßig auch das Serialisieren und Deserialisieren von *JavaBeans* (wiederverwendbare Softwarekomponenten, vgl. Sun Microsystems 1997) in XML-Dateien unterstützt (Sun Microsystems 2002a). Die im Paket `java.beans` definierten Klassen `XMLEncoder` und `XMLDecoder` stellen hierzu eine entsprechende `writeObject(Object)`- bzw. `readObject(Object)`-Methode bereit. Wird der XML-basierte Ausgabestrom in eine Datei umgeleitet, kann diese beispielsweise mit Hilfe der Java API for XML Processing (JAXP, vgl. Java Community Process 2006e) weiter bearbeitet werden. Als potentieller Nachteil der XML-Serialisierung sind jedoch die in der Regel längere Ausführungsdauer beim Schreiben und Lesen des Stroms sowie die größere Speicherallokation aufgrund der in XML üblichen strukturierenden und beschreibenden Elemente zu nennen. Weitere Ansätze, wie z. B. das Simple XML Serialization-API (vgl. http://simple.sourceforge.net/) bieten zusätzliche Konfigurationsmöglichkeiten etwa auf der Basis von Annotations und eine höhere Performance, setzen jedoch das Einbinden von zumindest einem zusätzlichen Java-Archiv voraus.

Im Umfeld kommerzieller Anwendungssysteme eignet sich die Java Objektserialisierung in der Regel nicht als Persistenzlösung. Die aus den wirtschaftlichen Anforderungen

ableitbaren technischen Erfordernisse, wie z. B. Transaktionskontrolle, Ad-hoc-Anfragen oder kurze Zugriffszeiten, lassen sich mit Hilfe der Objektserialisierung nicht direkt umsetzen. So ist beispielsweise ein direkter Zugriff auf ein einzelnes Objektattribut im Bytestrom weder zum Lesen noch zum Ändern möglich. Ist eine solche Änderung erforderlich, muss der gesamte Bytestrom gelesen und der vollständige Objektgraph im Hauptspeicher rekonstruiert werden. Nach erfolgter Modifikation muss dann wiederum der vollständige Objektgraph serialisiert werden. Weitere Probleme der Objektserialisierung ergeben sich aus der fehlenden Berücksichtigung statischer Klassenvariablen und einer möglichen Veränderung der Semantik des Domänen-Modells (vgl. Atkinson 2001). Zudem erweist sich häufig auch die fehlende Mehrbenutzersynchronisation als problematisch. Daher wird zur Umsetzung von Objektpersistenz häufig auf Datenbankmanagementsysteme zurückgegriffen, die über entsprechende Funktionalitäten verfügen.

3.4.2 Objektpersistenz und relationale Datenbankmanagementsysteme

Die in der Praxis stark verbreiteten relationalen Datenbanken sind zur Realisierung von Objekt-Persistenz aufgrund der unterschiedlichen Konzepte (relationales Datenmodell vs. objektorientiertem Paradigma), wie bereits in Abschnitt 3.1 deutlich wurde, nur bedingt geeignet (vgl. *Impedance Mismatch* in Schäfer 2003 oder auch Chen und Huang 1995). Zwar lassen sich generell Klassen als Relationen, Attribute als Spalten und Objekte als Tupel in einem relationalen Datenbankschema abbilden. Die Umsetzung objektorientierter Konzepte wie Kapselung, Polymorphismus, Vererbung und Beziehungen bereiten dagegen größere Probleme. Diese sollen am Beispiel und der Frage, wie sich eine objektorientierte Vererbungshierarchie in ein relationales Datenbankmanagementsystem abbilden lässt, exemplarisch aufgezeigt werden.

Bei der typisierten Partitionierung wird beispielsweise die gesamte Vererbungshierarchie auf eine Datenbanktabelle abgebildet. Diese Tabelle enthält für sämtliche Attribute der an der Vererbungshierarchie teilnehmenden Klassen eine korrespondierende Spalte. Zusätzlich schließt die Tabelle eine Typisierungsspalte mit ein. Die Typisierungsspalte beinhaltet die Information, zu welcher konkreten Klasse der Datensatz gehört. Ein möglicher Nachteil dieses Ansatzes ist, dass generell nur ein Teil der Spalten mit Daten belegt ist und die Tabelle somit viele Null-Werte aufweist. Zudem lassen sich auf Datenbankseite in diesem Fall keine „Not Null Constraints“ realisieren. Bei der horizontalen Partitionierung werden nur Subklassen auf der niedrigsten Ebene der Vererbungshierarchie in einer eigenständigen Datenbanktabelle abgebildet. Somit umfasst jede Tabelle neben den Attributen einer Subklasse auch die Attribute der Superklasse. Als nachteilig erweist sich in diesem Fall, dass die Attribute der Superklasse auf der untersten Vererbungsstufe redundant in mehreren Tabellen gespeichert werden müssen, was zu einem höherem Speicherplatzbedarf führt. Hinzu kommen potentielle Probleme bei Modifikationen an der Superklasse. Für Anfragen auf der obersten Ebene sind zudem Verbundoperationen über die gesamte Vererbungshierarchie notwendig, die zu einer hohen Last und langen Laufzeit führen können. Bei der vertikalen Partitionierung wird schließlich jede Klasse in einer eigenen Datenbanktabelle abgebildet. Für Anfragen, die sich auf eine Subklasse beziehen, sind somit Verbundoperationen notwendig, da die Attribute einer Instanz auf mehreren Tabellen verteilt sind. Je tiefer die Vererbungshierarchie, desto ineffizienter ist dieser Ansatz.

Aufgrund der hohen Komplexität wird insbesondere beim Einsatz objektorientierter Programmiersprachen und relationaler Datenbankmanagementsysteme von der Eigenentwicklung eines Persistenz-Frameworks abgeraten. Dennoch kommt es in der Praxis immer wieder dazu, dass Softwarearchitekten und Anwendungsentwickler den Zeit- und Kostenaufwand zur Konzeption und Implementierung eigener Persistenzlösungen unterschätzen. Dittert beanstandet diesen Sachverhalt und unterstreicht, dass „[...] die Neuentwicklung von Persistenz-Frameworks [...]“ eine komplexe Aufgabe darstellt, „[...] an der sich schon zahlreiche Entwicklungsteams verschlissen haben“ (2004, S. 36). Ursache für dieses Fehlverhalten sei der häufige Wunsch von Entwicklern, spannende und anspruchsvolle Aufgaben elegant lösen zu wollen, sowie eine Unterschätzung des Entwicklungs-, Test- und Wartungsaufwands. Eine derartige Vorgehensweise verursacht im Allgemeinen höhere Kosten als der Einsatz bestehender Softwareprodukte, da neben den einmaligen Entwicklungskosten auch Kosten für die Wartung und weitere Anpassungen angesetzt werden müssen (Dittert 2004).

Hinsichtlich des skizzierten Zeit- und Kostenaufwands, ist der Einsatz eines am Markt etablierten objektrelationalen Mapping-Werkzeugs einer Eigenentwicklung vorzuziehen. Der Begriff *Object/Relational Mapping* bezeichnet nach Bauer und King „[...] the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database“ (2006, S. 25). Ein so genannter O/R-Mapper verdeckt demnach den *Low-Level* Zugriff auf ein relationales Datenbankmanagementsystem und sorgt aus Sicht eines Anwendungsentwicklers für eine automatische und transparente Abbildung der Objekte. Durch dessen Einsatz entfällt die aufwendige Eigenentwicklung, und die Wartbarkeit der Kernanwendung steigt. Zwar kann eine selbsterstellte Persistenzlösung, die auf einen speziellen Anwendungsfall ausgerichtet ist, u. U. leistungsfähiger sein. Jedoch sollten in die Entscheidung für oder gegen eine Eigenentwicklung auch Aspekte der Budget- und Zeitplanung sowie der bereits angesprochene Entwicklungs-, Test- und Wartungsaufwand einbezogen werden. Mit Verabschiedung der Java Persistence API wird der Versuch unternommen, einen Standard zur objektrelationalen Abbildung zu etablieren (Java Community Process 2006b). Bisher ging mit der Verwendung eines entsprechenden Produkts, wie z. B. TopLink oder Hibernate, häufig auch die Bindung an eine konkrete Datenbank, eine spezifische Programmiersprache oder an das Mapping-Tool selbst einher. Auch die JDO-Spezifikation definiert einen einheitlichen Standard zur objektrelationalen Abbildung, gestattet aber überdies auch die Anbindung alternativer Datenbank- und Speichertechnologien (Java Community Process 2006h). In der Literatur werden diesbezüglich Migrationsstrategien untersucht und z. B. Vorgehensweisen zur Ablösung einer auf Hibernate basierenden Anwendung durch Einsatz einer JDO-Implementierung erörtert (vgl. z. B. Beeger, Haase, Roock und Sanitz 2006, S. 327-334).

3.4.3 Objektpersistenz und objektorientierte Datenbankmanagementsysteme

Aufgrund der konzeptuellen Unterschiede zwischen Objektmodell und relationalem Datenmodell propagierten die Autoren des *Object-Oriented Database System Manifesto* bereits 1989 die Verwendung objektorientierter Datenbankmanagementsysteme zur Realisierung von Objektpersistenz (Atkinson, Bancilhon, DeWitt, Dittrich, Maier und Zdonik 1989).

Sie begründeten die nach ihrer Ansicht gegebene Vorteilhaftigkeit eines OODBMS gegenüber einem RDBMS u. a. mit der konsequenten Anwendung des Objektmodells in Programmiersprache und Datenbankmanagementsystem.

Die ersten OODBMS ermöglichten bereits die transparente Persistenzierung komplex strukturierter Objekte und verfügten u. a. über Methoden für den navigierenden Zugriff auf diese. Das Fehlen übergreifender Standards führte jedoch zu gravierenden Unterschieden einzelner OODBMS bezüglich des zugrundeliegenden Objektmodells und der jeweils verwendeten Datendefinitions- und Anfragesprachen. Dieses Defizit versuchte die 1991 gegründete Object Database Management Group (ODMG) als Zusammenschluss führender OODBMS-Hersteller auszugleichen (Object Data Management Group 2000). Der erstmals 1993 verabschiedete ODMG-Standard sollte entsprechend eine herstellerübergreifende Datenhaltung für objektorientierte Datenbanken ermöglichen. Die im Jahr 2000 verabschiedete Version 3.0 des ODMG-Standards umfasst ein sprachneutrales Objektmodell, eine Objektspezifikationssprache, ein Objektaustauschformat, die deklarative Anfragesprache OQL (Object Query Language) sowie eigenständige Anbindungen für unterschiedliche Programmiersprachen (für die C++-Sprachanbindung vgl. z. B. Schader 1997). Obgleich der ODMG-Standard zu dieser Zeit von mehreren Herstellern implementiert wurde, konnten sich objektorientierte Datenbanken, entgegen der in der Literatur festgestellten Vorteilhaftigkeit, damals nicht am Markt durchsetzen.

Die ODMG stellte daher die Weiterentwicklung des ODMG-Standards ein und löste sich im Jahr 2001 auf. Lediglich die Java-Sprachanbindung wurde dem Java Community Process (JCP) als Ausgangsbasis zur Erarbeitung eines neuen Persistenz-Standards für Java übergeben (siehe Java Data Objects in Abschnitt 4.2). Allerdings stellen einige Autoren neuerdings auch wieder eine steigende Verbreitung objektorientierter Datenbanken fest und belegen dies am Beispiel des proprietären Produkts db4o, das heute u. a. in Firmen wie Boeing, Hertz und Bosch zum Einsatz kommt (Paterson, Edlich, Hörning und Hörning 2006, S. 13). Edlich geht diesbezüglich einen Schritt weiter und sieht insbesondere im Bereich von *Embedded Systems* bereits erste Anzeichen für eine „Renaissance der Objektdatenbanken“ gegeben (2006).

3.5 Fazit

Wie im vorherigen Abschnitt dargelegt wurde, existieren zahlreiche Ansätze und Produkte zur Realisierung transparenter Persistenz in Java. Neben der hier standardmäßig verfügbaren Objektserialisierung stehen Anwendungsentwicklern, u. a. die am Markt existierenden O/R-Mapper sowie objektorientierte Datenbanken zur Verfügung. In der Regel unterstützen die vorhandenen Persistenzlösungen jedoch lediglich eine kleine Auswahl bestimmter Datenbank- bzw. Speichertechnologien. Zudem unterscheiden sich diese häufig hinsichtlich der bereitgestellten API und der Syntax bzw. Mächtigkeit der Anfragesprache. Aufgrund der genannten Einschränkungen wurde innerhalb des Java Community Process (siehe Abschnitt 4.2) die Entwicklung einer einfachen, einheitlichen und transparenten Persistenzlösung (Java Data Objects-Spezifikation) angestoßen (Java Community Process 2004, S. 20), auf die das nachfolgende Kapitel ausführlich eingeht.

Literaturverzeichnis

Almaer, Dion (2003): *JDO 2.0: Lot's of Changes Discussed at the Kickoff Meeting, TheServerSide.COM,* 2003, http://www.theserverside.com/tt/articles/content/JDO2-Kickoff/article.html (abgerufen am 05.02.2007).

American National Standard for Information Technology (2003): *Information technology - Database languages - SQL - Part 10: Object Language Bindings (SQL/OLB),* 2003.

Anderson, Anne (2002): *Java Access Control Mechanisms,* Technical Report TR-2002-108, Sun Microsystems and Harvard University, 2002, http://research.sun.com/techrep/2002/smli_tr-2002-108.pdf (abgerufen am 05.02.2007).

ANSI, American National Standard for Information Technology (2004): *Role Based Access Control,* ANSI INCITS 359-2004, 2004.

Atkinson, Malcolm P. (1991): *A Vision of Persistent Systems.* In: C. Deloble; M. Kifer und Y. Manunaga (Hrsg.): Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases (DOOD), Lecture Notes in Computer Science, Munich, Germany, 1991, S. 453–459.

Atkinson, Malcolm P. (2001): *Persistence and Java - A Balancing Act.* In: Proceedings of the International Symposium on Objects and Databases, Springer-Verlag, London, UK, 2001, S. 1–31.

Atkinson, Malcolm P.; Bailey, Peter J.; Chisholm, Kenneth; Cockshott, W. Paul und Morrison, Ronald (1983): *An Approach to Persistent Programming, Computer Journal*, 1983, *26* (4), S. 360–365.

Atkinson, Malcolm P.; Bancilhon, François; DeWitt, D.; Dittrich, Klaus; Maier, David und Zdonik, Stanley (1989): *The Object-Oriented Database System Manifesto.* In: Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), North-Holland/Elsevier Science Publishers, Kyoto, Japan, 1989, S. 223–240.

Atkinson, Malcolm P.; Jordan, Mick; Daynès, L. und Spence, S. (1996): *Design Issues for Persistent Java: A Type-Safe, Object-Oriented, Orthogonally Persistent System.* In: Proceedings of the 7^{th} Workshop on Persistent Object Systems (POS'96), Cape May (NJ), USA, 1996, S. 33–47.

Atkinson, Malcolm P. und Jordan, Mick (2000a): *A Review of the Rationale and Architectures of PJama: A Durable, Flexible, Evolvable and Scalable Orthogonally Persistent Programming Platform,* 2000, http://research.sun.com/forest/COM.Sun.Labs.Forest.doc.pjama_review.review_pdf.pdf (abgerufen am 05.02.2007).

Atkinson, Malcolm P. und Jordan, Mick (2000b): *Orthogonal Persistence for the Java Platform: Specification and Rationale,* Technical Report, Mountain View, CA, USA 2000, http://research.sun.com/techrep/2000/abstract-94.html (abgerufen am 05.02.2007).

Atkinson, Malcolm P. und Morrison, Ronald (1995): *Orthogonally persistent systems, The VLDB Journal*, 1995, *4*, S. 319–402.

Azzi, Michael (2003): *A Criticism of Java Data Objects (JDO),* TheServerSide.com, Java Discussion Website, 2003, http://www.theserverside.com/news/thread.tss?thread_id=8571 (abgerufen am 05.02.2007).

Baer, Rudolf und Zängler, Pius (2000): *Wie misst man IT-Sicherheit?,* in: Heinz Sauerburger (Hrsg.): Praxis der Wirtschaftsinformatik – Security Management, Vol. HMD 216, Dpunkt-Verlag 2000, S. 67-77, 2000.

Bauer, Christian und King, Gavin (2006): *Java Persistence with Hibernate – Second Edition of Hibernate in Action*, Greenwich, CT, USA, Manning Publications Co., 2006.

Bea Systems (2006): *BEA Kodo 4.1 Documentation,* BEA Kodo 4.1 Documentation Webseite, 2006, http://edocs.bea.com/kodo/docs41/full/pdf/kododev.pdf (abgerufen am 05.02.2007).

BEA Systems, Inc. und IBM Corp. (2005): *Service Data Objects,* 2005, http://ftpna2.bea.com/pub/downloads/Commonj-SDO-Specification-v2.0.pdf (abgerufen am 05.02.2007).

Beeger, Robert; Haase, Arno; Roock, Stefan und Sanitz, Sebastian (2006): *Hibernate - Persistenz in Java-Systemen mit Hibernate 3*, 1. Auflage, Heidelberg, Dpunkt-Verlag, 2006.

Biggs, Maggie (2002): *Java Data Objects: Standard, Simplified Access to Persistent Data,* Online-Artikel, DevX, 2002, http://archive.devx.com/java/free/articles/jdobiggs/jdob100901-1.asp (abgerufen am 05.02.2007).

Blosser, Jeremy (2000): *Explore the Dynamic Proxy API,* 2000, http://java.sun.com/developer/technicalArticles/DataTypes/proxy/ (abgerufen am 05.02.2007).

Bode, Arndt und Hellwagner, Hermann (2006): *Leistungsbewertung.* In: Peter Rechenberg und Gustav Pomberger (Hrsg.): Informatik Handbuch, 4. Auflage, München und Wien, Carl Hanser Verlag, 2006, S. 453–470.

Bundesamt für Sicherheit in der Informationstechnik (2005): *E-Government-Handbuch, Unterkapitel IV B: IT und IT-Sicherheit, Authentisierung im E-Government,* 2005, http://www.bsi.bund.de/fachthem/egov/download/4_Authen.pdf (abgerufen am 05.02.2007).

Bundesamt für Sicherheit in der Informationstechnik (2006): *IT-Grundschutz-Kataloge,* 2006, http://www.bsi.bund.de/gshb/deutsch/download/ (abgerufen am 05.02.2007).

Cabri, Giacomo; Ferrari, Luca und Leonardi, Letizia (2006): *Applying security policies through agent roles: a JAAS based approach, Science of Computer Programming*, 2006, *59* (1-2), S. 127–146.

Chen, Jian und Huang, Qiming (1995): *Eliminating the Impedance Mismatch Between Relational Systems and Object-Oriented Programming Languages,* 1995, http://citeseer.ist.psu.edu/cache/papers/cs/155/http:zSzzSzinsect.sd.monash.edu.auzSzresearchzSzpublicationszSz1995zSzP95-4.pdf/chen95eliminating.pdf (abgerufen am 05.02.2007).

Curtin, Matt (1998): *Write Once, Run Anywhere: Why It Matters,* Online-Artikel, 1998, http://www.interhack.net/people/cmcurtin/rants/write-once-run-anywhere/ (abgerufen am 05.02.2007).

De Win, Bart; Piessens, Frank; Joosen, Wouter und Verhanneman, Tine (2002): *On the importance of the separation-of-concerns principle in secure software engineering,* Workshop on the Application of Engineering Principles to System Security Design, Boston, MA, USA, November 6–8, 2002, http://www.cs.kuleuven.ac.be/~frank/PAPERS/WAEPSSD.pdf (abgerufen am 05.02.2007).

DeMichiel, Linda und Russell, Craig (2004): *A Letter to the Java Technology Community,* 2004, http://java.sun.com/j2ee/letter/persistence.html (abgerufen am 05.02.2007).

Department of Defense (1985): *Trusted Computer System Evaluation Criteria,* 1985, http://csrc.nist.gov/secpubs/rainbow/std001.txt (abgerufen am 05.02.2007).

Dierstein, Rüdiger (2004): *Sicherheit in der Informationstechnik – der Begriff IT-Sicherheit, Informatik-Spektrum*, 2004, *27, 4*, S. 343–353, http://www.springerlink.com/content/13b6x34xu34u9u3h (abgerufen am 05.02.2007).

Dittert, Kerstin (2004): *Softwarearchitektur: Mythen und Legenden, Objektspektrum*, 2004, *3*, S. 34–39, http://www.sigs.de/publications/os/2004/03/dittert_OS_03_04.pdf (abgerufen am 05.02.2007).

Eckert, C. (2005): *IT-Sicherheit – Konzepte - Verfahren - Protokolle*, 1. Auflage, München, Oldenbourg Wissenschaftsverlag, 2005.

Edlich, Stefan (2006): *Renaissance der Objektdatenbanken?,* Online-Artikel, Entwickler.de, Software & Support Verlag GmbH, 2006, http://entwickler.de/zonen/portale/psecom,id,101,online,824,p,0.html (abgerufen am 05.02.2007).

Ernst & Young (2005): *Global Information Security Survey 2005, Report on the Widening Gap,* 2005.

Ferraiolo, David F.; Sandhu, Ravi S.; Gavrila, Serban I.; Kuhn, D. Richard und Chandramouli, Ramaswamy (2001): *Proposed NIST standard for role-based access control, Information and System Security*, 2001, *4* (3), 224–274, http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf (abgerufen am 05.02.2007).

Ferraiolo, David und Kuhn, Richard (1992): *Role-Based Access Controls.* In: 15^{th} NIST-NCSC National Computer Security Conference, 1992, S. 554–563, http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf (abgerufen am 05.02.2007).

Freeman, Elisabeth; Freeman, Eric; Bates, Bert und Sierra, Kathy (2004): *Head First Design Patterns*, O'Reilly, 2004.

Fumy, Walter und Kessler, Volker (2006): *Kryptologie und Datensicherheit.* In: Peter Rechenberg und Gustav Pomberger (Hrsg.): Informatik Handbuch, 4. Auflage, München und Wien, Carl Hanser Verlag, 2006, S. 235–257.

Gallaher, M.P.; O'Connor, A.C. und Kropp, B. (2002): *The Economic Impact of Role-Based Access Control,* National Institute for Standards & Technology (NIST), Technology Administration, 2002, http://www.nist.gov/director/prog-ofc/report02-1.pdf (abgerufen am 05.02.2007).

Gamma, Erich; Helm, Richard; Johnson, Ralph und Vlissides, John (1995): *Design Patterns, Elements of Reusable Object-Oriented Software*, 1st, Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 1995.

Göbel, Andreas (2005): *Rechtliche Risiken fehlender IT-Sicherheit, IT-Symposium 2005,* 2005, http://www.decus.de/slides/sy2005/05_04/1D04.pdf (abgerufen am 05.02.2007).

Glatschke, Christian (2004): *Der Java Community Process, Java Spektrum*, 2004, *04*, S. 68–69, http://www.sigs-datacom.de/sd/publications/pub_article_show.htm?&AID=1372&Table=sd_article/ (abgerufen am 17.11.2006).

Gong, Li (2002): *Java 2 Platform Security Architecture,* 2002, http://java.sun.com/javase/6/docs/technotes/guides/security/spec/security-spec.doc.html (abgerufen am 05.02.2007).

Gosling, James und McGilton, Henry (1996): *The Java Language Environment, White Paper,* 1996, http://java.sun.com/docs/white/langenv/ (abgerufen am 05.02.2007).

Grauer, Michael (2006): *Design und Implementation eines Systems zur benutzerbasierten Verwaltung von Java-Berechtigungen auf Basis des JDO-Standards,* Diplomarbeit, vorgelegt am Lehrstuhl für Wirtschaftsinformatik III, Universität Mannheim, 2006.

Heinrich, Lutz J. (2000): *Bedeutung von Evaluation und Evaluationsforschung in der Wirtschaftsinformatik.* In: Lutz J. Heinrich und Irene Häntschel (Hrsg.): Evaluation und Evaluationsforschung in der Wirtschaftsinformatik, Oldenbourg Wissenschaftsverlag, München, Wien, 2000, S. 7–22.

Heise Online (2005): *Erneut millionenfacher Diebstahl von Kreditkarten-Daten,* Newsticker, Heise Zeitschriften Verlag, 2005, http://www.heise.de/newsticker/meldung/58733 (abgerufen am 05.02.2007).

Heise Online (2006a): *Datenbank der Universität von Kalifornien gehackt,* Newsticker, Heise Zeitschriften Verlag, 2006, http://www.heise.de/newsticker/meldung/82437 (abgerufen am 05.02.2007).

Heise Online (2006b): *Massiver Diebstahl von Kundendaten bei AT&T,* Newsticker, Heise Zeitschriften Verlag, 2006, http://www.heise.de/newsticker/meldung/77455 (abgerufen am 05.02.2007).

Heuer, Andreas (1997): *Objektorientierte Datenbanken – Konzepte, Modelle, Standards und Systeme*, 2., aktualisierte und erweiterte Auflage, Bonn u. a., Addison-Wesley, 1997.

Hibernate (2006a): *Hibernate Reference Documentation, Version 3.2.2,* 2006, http://www.hibernate.org/hib_docs/v3/reference/en/pdf/hibernate_reference.pdf (abgerufen am 05.02.2007).

Hibernate (2006b): *Hibernate Security: Declarative permissions using JAAS and Interceptors,* 2006, http://www.hibernate.org/140.html (abgerufen am 05.02.2007).

Hoppe, Gabriela und Prieß, Andreas (2003): *Sicherheit von Informationssystemen – Gefahren, Maßnahmen und Management im IT-Bereich*, 1. Auflage, Herne, Berlin, NWB Verlag, 2003.

Java Community Process (2002a): *JSR-012: Java Data Objects (JDO) Specification, Maintenance Release, Version 1.0,* 2002, http://www.jcp.org/en/jsr/detail?id=12 (abgerufen am 05.02.2007).

Java Community Process (2002b): *JSR-055: Java Certification Path API, Final Release, Version 1.0,* 2002, http://www.jcp.org/en/jsr/detail?id=55 (abgerufen am 05.02.2007).

Java Community Process (2003a): *JSR-112: J2EE Connector Architecture 1.5,* 2003, http://www.jcp.org/en/jsr/detail?id=112 (abgerufen am 05.02.2007).

Java Community Process (2003b): *JSR-235: Service Data Objects,* 2003, http://www.jcp.org/en/jsr/detail?id=235 (abgerufen am 05.02.2007).

Java Community Process (2004): *JSR-012: Java Data Objects (JDO) Specification, Maintenance Release, Version 1.0.1,* 2004, http://www.jcp.org/en/jsr/detail?id=12 (abgerufen am 05.02.2007).

Java Community Process (2006a): *Introduction to the Java Community Process,* 2006, http://www.jcp.org/en/introduction/overview/ (abgerufen am 05.02.2007).

Java Community Process (2006b): *Java Persistence API, Final Release,* 2006, http://www.jcp.org/en/jsr/detail?id=220 (abgerufen am 05.02.2007).

Java Community Process (2006c): *JSR-115: Java Authorization Contract for Containers, Maintenance Release, Version 1.1,* 2006, http://www.jcp.org/en/jsr/detail?id=115 (abgerufen am 05.02.2007).

Java Community Process (2006d): *JSR-154: Servlet Specification, Maintenance Draft Review 6, Version 2.5,* 2006, http://www.jcp.org/en/jsr/detail?id=154 (abgerufen am 05.02.2007).

Java Community Process (2006e): *JSR-206: Java API for XML Processing (JAXP), Maintenance Release, Version 1.3,* 2006, http://www.jcp.org/en/jsr/detail?id=206 (abgerufen am 05.02.2007).

Java Community Process (2006f): *JSR-220: Enterprise JavaBeans 3.0, Final Release,* 2006, http://www.jcp.org/en/jsr/detail?id=220 (abgerufen am 05.02.2007).

Java Community Process (2006g): *JSR-221: JDBC 4.0 API Specification, Final Release, Version 4.0,* 2006, http://www.jcp.org/en/jsr/detail?id=221 (abgerufen am 05.02.2007).

Java Community Process (2006h): *JSR-243: Java Data Objects 2.0 - An Extension to the JDO specification, Final Release, Version 2.0,* 2006, http://www.jcp.org/en/jsr/detail?id=243 (abgerufen am 05.02.2007).

Java Community Process (2006i): *JSR-243: Java Data Objects 2.0 - An Extension to the JDO specification, Maintenance Draft Review, Version 2.1, November 20006,* 2006, http://jcp.org/aboutJava/communityprocess/maintenance/jsr243/index.html (abgerufen am 05.02.2007).

Java Persistent Objects (2006): *JPOX - Tutorial for JDO, JDO 2.0 Referenzimplementierung,* 2006, http://www.jpox.org/docs/1_2/tutorials/jdo_tutorial.html (abgerufen am 05.02.2007).

Jordan, David (2002a): *Comparing JDO and ODMG,* Online-Artikel, JDOcentral.com, Developer's Community for Java Persistence Standards, 2002, http://www.jdocentral.com/JDO_Commentary_DavidJordan_4.html (abgerufen am 05.02.2007).

Jordan, David (2002b): *A Comparison Between Java Data Objects (JDO), Serialization and JDBC for Java Persistence,* Online-Artikel, JDOcentral.com, Developer's Community for Java Persistence Standards, 2002, http://jdocentral.com/pdf/DavidJordan_JDOversion_12Mar02.pdf (abgerufen am 05.02.2007).

Jordan, David (2003): *JDO Gets a Boost from „Down Under",* Online-Artikel, JDOcentral.com, Developer's Community for Java Persistence Standards, 2003, http://www.jdocentral.com/JDO_Commentary_DavidJordan_11.html (abgerufen am 05.02.2007).

Jordan, David und Russell, Craig (2003a): *Java Data Objects – Store Objects with Ease*, Sebastopol, CA, USA, O'Reilly, 2003.

Jordan, David und Russell, Craig (2003b): *JDO or CMP?,* Online-Artikel auf ONJava.com, O'Reilly, 2003, http://www.onjava.com/pub/a/onjava/2003/05/21/jdo.html (abgerufen am 05.02.2007).

Jordan, Mick (2004): *A Comparative Study of Persistence Mechanisms for the Java Platform, Sun Microsystem,* 2004, http://research.sun.com/techrep/2004/smli_tr-2004-136.pdf (abgerufen am 05.02.2007).

Kappel, Gerti und Schröder, Birgit (1998): *Distributed Light-Weight Persistence in Java - A Tour on RMI- and CORBA-Based Solutions.* In: Database and Expert Systems Applications, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, Germany, 1998, S. 411–424, ftp://ftp.ifs.uni-linz.ac.at/pub/publications/1998/0898.pdf (abgerufen am 05.02.2007).

Korb, William (2004): *Using JDO for Transparent Persistence,* Online-Artikel, JavaPro, 2004, http://www.ftponline.com/javapro/2004_12/online/wkorb_12_08_04/ (abgerufen am 05.02.2007).

Li, Ninghui; Byun, Ji-won und Bertino, Elisa (2006): *A Critique of the ANSI Standard on Role Based Access Control,* Purdue University, CERIAS and Department of Computer Science, Beitrag zur Begutachtung eingereicht bei IEEE Security and Privacy, 2006, http://www.cs.purdue.edu/homes/ninghui/papers/aboutRBACStandard.pdf (abgerufen am 05.02.2007).

Lopes, Cristina V. und Hursch, Walter L. (1995): *Separation of Concerns,* College of Computer Science, Northeastern University, Boston, 1995, ftp://ftp.ccs.neu.edu/pub/people/crista/publications/techrep95/separation.pdf (abgerufen am 05.02.2007).

Marquez, Alonso; Blackburn, Stephen; Mercer, Gavin und Zigman, John N. (2001): *Implementing Orthogonally Persistent Java.* In: POS-9: Revised Papers from the 9^{th} International Workshop on Persistent Object Systems, Springer-Verlag, London, UK, 2001, S. 247–261, http://www.springerlink.com/content/95jhmdfp3yjq580m/ (abgerufen am 05.02.2007).

Meier, Andreas und Wüst, Thomas (1997): *Objektorientierte Datenbanken – Ein Kompaß für die Praxis*, Heidelberg, Dpunkt-Verlag, 1997.

Merz, Matthias (2006a): *JDOSecure: A Security Architecture for the Java Data Objects-Specification,* Proceedings of the 15^{th} International Conference on Software Engineering and Data Engineering (SEDE-2006), 6.-8. July, Los Angeles, California, USA, S. 134–140, 2006.

Merz, Matthias (2006b): *JDOSecure: Eine Sicherheitsarchitektur für die Java Data Objects-Spezifikation,* Frühjahrstreffen der GI-Fachgruppe Datenbanken zum Themenbereich Informationssicherheit und Datenschutz, 6. und 7. April, Hamburg, 2006.

Merz, Matthias (2006c): *The Management of Users, Roles, and Permissions in JDOSecure,* Proceedings of the International Conference on Principles and Practices of Programming in Java (PPPJ), Mannheim, Germany, S. 85–93, 2006.

Merz, Matthias (2006d): *Using the Dynamic Proxy Approach to Introduce Role-Based Security to Java Data Objects,* Proceedings of the 18^{th} International Conference on Software Engineering and Knowledge Engineering (SEKE'06), Skokie, Illinois, USA, S. 404–409, 2006.

Merz, Matthias und Aleksy, Markus (2006): *Using JDOSecure to Introduce Role-Based Permissions to Java Data Objects-based Applications,* Proceedings of the 17^{th} International Conference on Database and Expert Systems Applications, Krakow, Poland, S. 449–458, 2006.

Merz, Matthias und Grauer, Michael (2006): *Einführung eines Managementsystems zur Verwaltung von Benutzern, Rollen und Rechten in JDOSecure,* Lehrstuhl für Wirtschaftsinformatik III, Universität Mannheim, Diskussionspapier 1-06, 2006.

Merz, Matthias und Korthaus, Axel (2003): *A Critical Analysis of JDO in the Context of J2EE.* In: Al-Ani Ban; H. Arabnia und Mun Youngsong (Hrsg.): International Conference on Software Engineering Research and Practice (SERP '03), Vol. I CSREA 2003, S. 34–40.

Mohr, K.-L. (1993): *Art der Bedrohung.* In: Hartmut Pohl und Gerhard Weck (Hrsg.): Einführung in die Informationssicherheit, München u. a., Oldenbourg Wissenschaftsverlag, 1993, S. 33–34.

Munro, David. S. (1993): *On the Integration of Concurrency, Distribution and Persistence, Ph.D. Thesis*, University of St Andrews, Department of Mathematical and Computational Sciences, 1993.

Neuman, C.; Yu, T.; Hartman, S. und Raeburn, K. (2005): *The Kerberos Network Authentication Service (V5),* RFC 4120 (Proposed Standard), 2005, http://www.ietf.org/rfc/rfc4120.txt (abgerufen am 05.02.2007).

Oaks, Scott (2001): *Java Security* The Java Series, Second, Sebastopol, CA, USA, O'Reilly & Associates, Inc., 2001.

Object Data Management Group (2000): *ODMG 3.0,* 2000, http://www.odmg.org/ (abgerufen am 05.02.2007).

Paterson, Jim; Edlich, Stefan; Hörning, Henrik und Hörning, Reidar (2006): *The Definitive Guide to db4o*, Berkely, CA, USA, Apress, 2006.

Pieprzyk, Josef; Hardjono, Thomas und Seberry, Jennifer (2003): *Fundamentals of computer security*, Berlin, Heidelberg, u. a., Springer, 2003.

Pohl, Hartmut (2004): *Taxonomie und Modellbildung in der Informationssicherheit.* In: Datenschutz und Datensicherheit, Vol. 28, 11, Vieweg, 2004, S. 678–685.

Reese, George (2003): *Java Database Best Practices – Persistence Models and Techniques for Java Database Programming*, 1. ed., O'Reilly, 2003.

Richardson, Chris (2006): *POJOs in Action: Developing Enterprise Applications with Lightweight Frameworks*, Greenwich, CT, USA, Manning Publications Co., 2006.

Roos, Robin (2002): *Java Data Objects*, 1st, London u. a., Addison-Wesley, Pearson Education, 2002.

Rosenberger, Carl (2001): *Why Sun's Java Data Objects (JDO) will not be successful,* Online-Diskussionsforum, 2001, http://www.javalobby.org/forums/thread.jspa?forumID=46\&threadID=1326 (abgerufen am 05.02.2007).

Russell, Craig (2001): *Craig Russell Responds to Roger Sessions' Critique of JDO,* TheServerSide.com, Java Discussion Website, 2001, http://www.theserverside.com/articles/article.tss?l=RusselvsSessions (abgerufen am 05.02.2007).

Saake, Gunter; Schmitt, Ingo und Türker, Can (2003): *Objektdatenbanken – Konzepte, Sprachen, Architekturen*, aktualisierte PDF-Version der gebundenen Ausgabe, International Thomson Publishing, 2003.

Saake, Gunter und Sattler, Kai-Uwe (2003): *Datenbanken und Java, JDBC, SQLJ, ODMG und JDO*, 2., überarbeitete und aktualisierte Auflage, Heidelberg, Dpunkt-Verlag, 2003.

Saltzer, J. und Schroeder, M. (1975): *The Protection of Information in Computer Systems, Proceedings IEEE 63*, 1975, *9*, S. 1287–1308.

Samar, Vipin und Lai, Charlie (1996): *Making Login Services Independent of Authentication Technologies,* 1996, http://www.sun.com/software/solaris/pam/pam.external.pdf (abgerufen am 05.02.2007).

Samar, Vipin und Schemers, Roland (1996): *RFC 86.0: Unified Login with Pluggable Authentication Modules (PAM),* Open Software Foundation, Request For Comments 86.0, 1996, http://www.opengroup.org/tech/rfc/mirror-rfc/rfc86.0.txt (abgerufen am 05.02.2007).

Sandhu, Ravi S.; Coyne, Edward J.; Feinstein, Hal L. und Youman, Charles E. (1996): *Role-Based Access Control Models, IEEE Computer*, 1996, *29* (2), 38–47.

Schader, Martin (1997): *Objektorientierte Datenbanken – Die C++-Anbindung des ODMG-Standards*, 1. Auflage, Berlin, Heidelberg u. a., Springer, 1997.

Schader, Martin und Schmidt-Thieme, Lars (2003): *Java – Eine Einführung*, 4., aktualisierte und erweiterte Auflage, Berlin, Heidelberg u. a., Springer, 2003.

Schäfer, Alexandra (2003): *Der „Impedance Mismatch": Im Spannungsfeld zwischen objektorientiertem und relationalem Ansatz, Objektspektrum*, 2003, *3*, S. 33–36, http://www.sigs.de/publications/os/2003/03/schafer_OS_03_03.pdf (abgerufen am 05.02.2007).

Schlienger, Thomas (2006): *Einfluss von Mitarbeitenden auf die Informationssicherheit, Fachzeitschrift für Elektrotechnik, Bulletin SEV/VSE*, 2006, *Bulletin-Nummer 0607*, S. 27–30.

Schmidt, Duri (1991): *Persistente Objekte und Objektorientierte Datenbanken*, München, Wien, Hanser-Verlag, 1991.

Seiler, Martin (2006): *Studie: Stellenwert der IT-Sicherheit steigt, Computerwoche, IDG Business Verlag GmbH, München*, 2006, (42), S. 12.

Sun Microsystems (1997): *JavaBeans API Specification, Version 1.01,* 1997, http://java.sun.com/products/javabeans/docs/spec.html (abgerufen am 05.02.2007).

Sun Microsystems (2002a): *API Enhancements to the JavaBeans Component API in v1.4,* 2002, http://java.sun.com/j2se/1.4.2/docs/guide/beans/changes14.html (abgerufen am 05.02.2007).

Sun Microsystems (2002b): *Java Cryptography Architecture, API Specification & Reference,* 2002, http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html (abgerufen am 05.02.2007).

Sun Microsystems (2003): *Java Object Serialization Specification,* 2003, http://java.sun.com/j2se/1.5/pdf/serial-1.5.0.pdf (abgerufen am 05.02.2007).

Sun Microsystems (2005a): *Java Security Overview, White Paper,* 2005, http://java.sun.com/developer/technicalArticles/Security/whitepaper/JS_White_Paper.pdf (abgerufen am 05.02.2007).

Sun Microsystems (2005b): *The Java Language Specification*, 3rd, Addison-Wesley Professional, 2005, http://java.sun.com/docs/books/jls/ (abgerufen am 05.02.2007).

Sun Microsystems (2006a): *Java Authentication and Authorization Service (JAAS), Reference Guide for the Java SE Development Kit 6,* 2006, http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html (abgerufen am 05.02.2007).

Sun Microsystems (2006b): *Java Platform, Standard Edition 6 API Specification,* 2006, http://java.sun.com/javase/6/docs/api/ (abgerufen am 05.02.2007).

Sun Microsystems (2006c): *Java SE Security,* 2006, http://java.sun.com/javase/technologies/security/index.jsp (abgerufen am 05.02.2007).

Sun Microsystems (2007): *Core Java, Java Secure Socket Extension (JSSE),* 2007, http://java.sun.com/products/jsse/ (abgerufen am 05.02.2007).

The Open Group (1997): *X/Open Single Sign-on Service (XSSO) - Pluggable Authentication Modules,* Preliminary Specification, X/Open Document Number P702m, 1997, http://www.opengroup.org/onlinepubs/8329799/toc.pdf (abgerufen am 05.02.2007).

Tyagi, Sameer; Vorburger, Michael; McCammon, Keiron und Bobzin, Heiko (2004): *Core Java Data Objects*, 1st, Palo Alto, California, Sun Microsystems Press, Prentice Hall PTR, 2004.

Weimer, Uwe (2006): *Datenintegration - Der Schlüssel zum BI-Erfolg, ERP Management - Zeitschrift für unternehmesweite Anwendungssysteme, Gito mbH Verlag, Berlin*, 2006, *2* (3), S. 15–18.

Wende, Ingo und Roßnagel, Alexander (2006): *Normen und Recht.* In: Peter Rechenberg und Gustav Pomberger (Hrsg.): Informatik Handbuch, 4. Auflage, München und Wien, Carl Hanser Verlag, 2006, S. 1149–1192.