Daten als Objekte speichern mit db4o, eine objektorientierte Datenbank

Matthias Bordach
Universitaet Siegen
matthias.bordach@student.uni-siegen.de

ABSTRACT

Die Hausarbeit befasst sich mit der Datenbanktechnologie `db4o`. Das objektorientierte Speicherkonzept zeichnet sie als einen Vertreter der NO-SQL-Familie aus. Auf eine theoretische Einführung in den Standard ODMG folgt eine praktische Vorstellung der Datenbankfunktionalität. Die Funktionen werden mit Beispielen gestützt, die in Java geschrieben sind. Abfrage- und Speichermethoden werden ebenso erklärt wie die Realisierung von Beziehungen zwischen Objekten und das Handling von Schema Evolution. Der Leser wird nach der Lektüre in der Lage sein, eine einfache Datenbank mit db4o zu programmieren.

1. EINLEITUNG

Die Idee einer Datenbank, die Objekte ohne ein aufwendiges Mapping in ein anderes Datenmodell speichern kann, wurde erstmals in den 80er Jahren des letzten Jahrhunderts in konkreten Systemen umgesetzt. Die ersten objektorientierten Programmiersprachen etablierten sich in den Entwicklerkreisen. Man kapselte Rechenlogik und Zustände in eigene Umgebungen, sogenannte Klassen. Das Datenmodell einer Software besteht aus vielen solchen Klassendefinitionen. Objekte sind die erzeugten Instanzen dieser Schablonen und deren Zustände bilden den Gesamtzustand der Software zur Laufzeit und bei Bedarf dank Datenbanken persistent auch ausserhalb der Laufzeit. Relationale Datenbankensysteme wie die Vertreter aus der SQL-Familie speichern Daten in Tabellen ab. Dies erfordert Vorarbeiten für die Entwickler. Das Datenmodell der Software muss dementsprechend auf die Tabellen abgebildet werden. Die Structured Query Language ist eine Datenbanksprache mit eigenen Schlüsselwörtern und Syntax. Diese Tatsache stellt ebenfalls ein paralleler Entwicklungsaufwand dar, nämlich die Integration der Queries in die Programmiersprache. Diese zusätzlichen Arbeiten sind einer der Gründe für die Entwicklung alternativer Datenbanksysteme. Wie eine objektorientierte Datenbank eine einfache und effiziente Lösung liefert, wird in dieser Ausarbeitung vorgestellt.

2. DER ODMG STANDARD[1]

Verschiedene Ansätze für eine Datenbank auf objektorientierter Basis erforderten einen Standard. Diesen Standard entwickelte eine Arbeitsgruppe der Object Management Group (OMG). 1993 wurde die erste Version veröffentlicht mit dem Namen `The Object Data Standard ODMG`. Die Arbeitsgruppe überarbeitete und erweiterte das Dokument in den folgenden Jahren. Inhalt und Ziel des Dokuments ist eine Standarddefinition von Konzepten für die Anbindung eines objektorientierten Datenbankmanagementsystems an die gewählte Programmiersprache. In Version 3 wird den Sprachen C++, Smalltalk und Java jeweils ein eigenes Kapitel für die Schnittstellenumsetzung gewidmet. Generell definiert die ODMG das `Object Model`, Objektspezifikationssprachen und die an die SQL angelehnte `Object Query Language` (OQL). Es werden Standardausdrücke festgelegt für die Datenbankinstanziierung, -nutzung (open, close), Transaktionen (commit) und Mehrfachzugriff (Locks). In dem Objektmodell der ODMG wird das Konzept Objekt definiert. Es geht dabei ausführlich auf Objektstrukturkomponenten und -beziehungen ein. Die Speicherung der Objekte erfolgt entweder explizit (durch direkten Befehl) oder implizit (durch Referenzierung, Stichwort Extent). Abbildung 1 zeigt ein Beispiel eines Objektschemas. Vererbungen und Referenzierungen erzeugen eine komplexe Beziehungsstruktur.

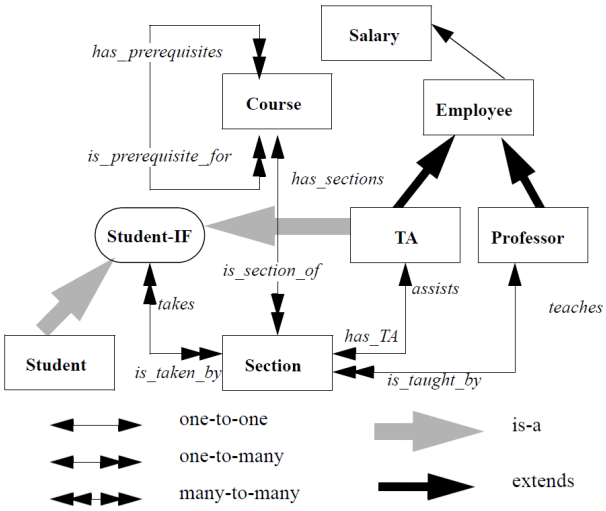

Figure 1: Graphische Repräsentation eines Objektschemas (Quelle ODMG 3.0 Kapitel 3.2.2)

3. DIE TECHNOLOGIE: DB4O

Db4o steht für Data Base for Objects. Die Aufgabe dieser Technologie ist, Objekte aus einer objektorientiert geschriebenen Software persistent zu speichern. In den ersten 2000er Jahren etablierte sich sowohl das kommerzielle als auch das freie Lizenzangebot der db4o-Anbieter. Bekannte Unternehmen wie IBM, Intel, Postbank, BOSCH Sigpack, BMW werden als Referenz aufgeführt. Die Zielgruppe sind Persistenzarchitekturen ohne aktive Datenbankadministration, vorzugsweise mobile und desktop clients.[2] Die Entwicklung profitiert auch von der Community, da das ganze Projekt als open-source deklariert ist. Im Jahre 2008 übernahm Versant Corporation das Produkt für 2.257.025 $[3] und vertrieb es unter diesem zweigleisigen Business-Model weiter neben ihren eigenen Datenbank-Angeboten. Actian Corporation übernahm Versant 2012 und stellte die Weiterentwicklung von db4o einige Zeit später ein. Es werden keine neuen kommerziellen Lizenzen vergeben, aber bestehende Verträge werden unterstützt. Für die freie Softwarenutzung steht die Datenbank weiterhin zur Verfügung. Db4o kann als Embedded- und als Client/Server-Variante genutzt werden. Gerade im mobilen Bereich eignet sich die Datenbank als Einbettung in eine reine Client-Software, da die Gesamtbibliothek nicht mehr als 1MB auf dem Festspeicher ausmacht.

3.1 Die Komponenten

Einen Link zum Download der aktuellen Version für Java oder .NET gibt es auf der Homepage von Actian[1]. Im Softwareumfang enthalten ist eine umfangreiche Dokumentation, die Quell- und Bibliotheksdateien sowie das db4o eigene Datenbankmanagementsystem `Object Manager Enterprise`. Die Aufteilung der Bibliotheksdateien bietet dem Anwender die Möglichkeit die benötigten Komponenten einzeln in ein Projekt einzubinden. Dabei stellen die folgenden Bibliotheken die wichtigsten dar:

- `db4o-8.0-core-java5.jar`
  Die Basisbibliothek für Java 5 und aufwärts.
- `db4o-8.0-optional-java5.jar`
  Eine Erweiterung der Grundfunktionalität.
- `db4o-8.0-cs-java5.jar` Notwendige Bibliothek für die client/server-Variante.
- `db4o-8.0-all-java5.jar`
  Die Gesamtfunktionalität der Datenbank in einer Datei. Sie ersetzt sämtliche Einzelbibliotheken.

3.2 Die Grundlagen

Ein gewöhnlicher Datenbankzugriff benötigt folgende Schritte:

1. Öffnen einer Datenbank mit Angabe einer Datei

```
EmbeddedConfiguration config =
  Db4oEmbedded.newConfiguration();
ObjectContainer db = Db4oEmbedded
  .openFile(config, "datenbank.yap");
```

2. Abfrage von Objekten und Sammeln der Ergebnisse in einem Set

```
ObjectSet result =
  db.queryByExample(new Object());
```

3. Verwenden und Verändern der Objekte

```
Object obj = result.next();
System.out.println(obj);
```

4. Speichern eines Objektes in die Datenbank

```
db.store(obj);
```

5. Löschen eines Objektes aus der Datenbank

```
db.delete(obj);
```

6. Schließen der Datenbank

```
db.close();
```

Die Schritte 1 und 6 sind verpflichtend. Alle Schritte zwischen 1 und 6 müssen in einem try-catch-Block eingebettet sein. Je nach Ziel des Datenbankzugriffes variieren natürlich die Schritte innerhalb des Blocks. Falls man zum Beispiel ein Objekt zum ersten Mal in der Datenbank speichern möchte, reicht Schritt 4 aus.
Ausgangspunkt ist das Objekt db4oEmbedded. Über die statische Methode openFile wird die gewünschte Datenbank mit einer modifizierbaren Konfiguration geladen. Der Rückgabewert dieser Methode ist eine Instanz der Klasse ObjectContainer. Dieses Interface kapselt die Datenbank und bietet die Zugriffsmethoden für sämtliche Basisfunktionalität. Mit dem Befehl `db.ext()` [2] erhält man die erweiterte Containerklasse mit mehr Funktionen.
Jedes Abfrageergebnis wird in einem ObjectSet gespeichert. Diese Kollektion ist wie gewönlich iterierbar. Die Elemente sind Objekte der Java-Basisklasse `java.lang.Object`. Sie müssen lediglich vor der Benutzung gecastet werden. Mit dem Befehl `store()` wird ein Objekt in der Datenbank gespeichert, falls es dort noch nicht existiert. Ansonsten mit dem aktuellen Zustand überschrieben. Mit `close()` wird die aktuelle Transaktion geschlossen.

3.3 Objektidentität

Die von der ODMG geforderte Objektidentität ist natürlich auch in db4o verwirklicht und wird folgendermaßen abgerufen bzw. aktiviert:

- ID
  Die Vergabe einer ID geschieht beim erstmaligen Speichern eines Objektes automatisch. Sie ist nur innerhalb dieser Datenbank eindeutig. Abrufbar ist sie folgendermaßen:

```
objectContainer.ext().getID(object);
```

- UUID
  Die Unique Universal ID markiert ein Objekt, sodass es beim Kopieren in eine andere Datenbank seine Identität behält. So können Datenbanken synchron gehalten werden. Das System vergibt aber aus Speicher- und Performancegründen keine UUIDs automatisch, sondern dieser Vorgang muss explizit aktiviert werden:

```
Db4o.configure().objectClass(Person.class)
  .generateUUIDs(true);
```

[1] http://supportservices.actian.com/versant/default.html

[2] db ist eine Instanz von ObjectContainer

3.4 Abfragemethoden

Anders als zum Beispiel bei relationalen Sprachen bietet db4o eine Abfragesprache nativ in Java an. Es muss kein Befehl ausgeführt werden, der einen String mit besonderen Abfragekonstrukten erfordert. Db4o stellt drei verschiedene Möglichkeiten zur Verfügung, wie Objekte gezielt oder ungefiltert aus der Datenbank geholt werden können. Dabei haben alle Varianten ihre Eigenheiten und Anwendungsvorteile. Es gibt die Query-by-Example-, Native Queries- und die SODA-Methode. Sie werden im Folgenden einzeln eingeführt.

3.4.1 Query by Example

Query by Example bedeutet auf Deutsch Abfrage durch Beispiel. Konzeptuell sieht diese Methode so aus, dass man der Datenbank eine Objektinstanz zeigt und die Datenbank alle Objekte des gleichen Typs herausgibt, die mit den Attributwerten des Musters übereinstimmen. Ein Java-Standardwert für einen Datentyp gilt dabei als wildcard, wird also bei der Prüfung durch die Datenbank ignoriert. Hat das Musterobjekt den Wert 0 bei einem Attribut des Typs int, so wird dieses Attribut bei der Objektfilterung ignoriert.

```
Person musterobjekt =
  new Person(null, "Meier", 34);
ObjectSet result =
  db.queryByExample(musterobjekt);
while (result.hasNext()) {
  Person p = (Person) result.next());
  System.out.println(p);
}
```

Dieses Beispiel erzeugt zuerst ein Objekt der Klasse Person. Der Vorname erhält den Wert null, Nachname und Alter sind fest. In dem db4o eigenen von java.util.List erbenden ObjectSet liefert der ObjectContainer db nun alle Personobjekte zurück, die mit Nachnamen Meier heißen und 34 Jahre alt sind. Da diese Objekte in dem Set noch als `Object` getypt sind, müssen sie bei Verwendung noch auf den Typ `Person` gecastet werden. Das Query by Example-Verfahren ist kompakt und schnell zu verstehen. Für komplexe Abfragen ist es eher ungeeignet, da es nur die Filterung auf einen bestimmten Attributwert erlaubt. Also eine gleichzeitige Abfrage auf Meier und Becker ist nicht möglich. Außerdem werden wie schon beschrieben Standardwerte ignoriert, was im obigen Beispiel dazu führt, dass im Falle einer 0 als Alter die Datenbank alle Meiers liefert, anstatt nur die Meiers die 0 Jahre alt sind.

3.4.2 Native Queries

Das Konzept der Native Queries basiert auf zwei Dokumente[6][7], an denen u.a. der db4object-Gründer Carl Rosenberger mitgewirkt hat. Die Dokumente definieren syntaktische Abfrageelemente bezüglich Zugriffe auf Objekte in Datenbanken und wie sie in objektorientierte Programmiersprachen eingebettet werden können. Diese Konstrukte werden den relationalen Befehlen gegenübergestellt und die Vorzüge von Native Querys hervorgehoben. Wie sie in db4o mit Java umgesetzt sind, demonstriert folgendes Beispiel:

```
Predicate predicate = new Predicate() {
  @Override
  public boolean match(Object obj) {
    return ((Person) obj).getNachname()
              .equals("Meier") ||
           ((Person) obj).getNachname()
              .equals("Becker"));
}};
ObjectSet result = db.query(predicate);
```

Die Methode, die an dem `ObjectContainer` db ausgeführt werden muss, heißt `query()`. Was den Befehl mächtig macht, ist das zu übergebende Argument. Das Argument ist eine Instanz der db4o eigenen Klasse Predicate. Bei der Instanziierung muss die Methode `match()` implementiert werden. Dieser Vorgang ähnelt dem mit dem Interface Comparator, bei dem die Methode `compare()` überschrieben werden muss. In dieser Methode können jetzt unterschiedlich komplexe Vergleichsoperationen vorkommen.
Der Mehrwert gegenüber Query By Example liegt auf der Hand. Jetzt ist es möglich, alle Personen aus der Datenbank zu holen, die entweder Meier oder Becker heißen. Diese Abfragemethode ist sehr rechenaufwendig.

3.4.3 SODA

SODA steht für Simple Object Database Access. Native Queries nutzt diesen Mechanismus implizit bei der Ausführung der Abfragen. Vereinzelte Syntaxelemente erinnern entfernt an einen SQL-Query. Am besten erklärt man diese Abfragevariante anhand eines Beispiels:

```
Query query = db.query();
query.constrain(Person.class);
Constraint c_name =
  query.descend("name").constrain("Meier").or(
  query.descend("name").constrain("Becker"));
query.descend("age").constrain(34).and(c_name);
ObjectSet result = query.execute();
```

Zuerst wird ein `Query` Objekt erzeugt. In diesem Objekt sammelt man alle Bedingungen, sogenannte Constraints, die für die Abfrage notwendig sind. Die interne Struktur eines Queries ähnelt einem Graphen. Die Knoten stellen die Suchbedingungen dar. Bei der Ausführung des Queries geht der Algorithmus die Knoten durch und filtert die Objekte, die auf die Bedingungen zutreffen. Im Beispiel ist der erste Constraint die Klasse Person. Dadurch wird die Ergebnismenge auf die Objekte mit dem komplexen Datentyp Person eingeschränkt. Die folgenden Constraints, angeführt mit der Methode `descend()`, sind Abfragen auf Attribute der Klasse Person. Descend meint dabei, durchlaufe die Objektstruktur, suche das Attribut mittels gegebenem String (name oder age) und teste es auf den gegebenen Wert (Meier oder 34). Die Einteilung der ersten beiden descend-Ausführungen in eine Variable ist eine Möglichkeit, um den Code zu strukturieren. Durch die Methoden `or()` und `and()` können komplexere Queries nach üblichen booleschen Regeln gestaltet werden. Am Schluß führt die Methode `execute()` auf das Query-Objekt die gesamte Abfrage und Objektfilterung aus und liefert wie gewohnt ein ObjectSet zurück.
Die Entwickler empfehlen die überwiegende Nutzung der Native Queries. Aber mitunter macht es auch Sinn, auf die SODA-Variante zurückzugreifen, da sie dynamischer konstruiert werden kann und die Abfragen effizientere Laufzeit aufweisen. Der Nachteil ist aber, dass Fehleingaben bei den descend-Konstrukten nicht zur Kompilierzeit abgefangen werden und die Syntax aufwendiger ist.

3.5 Beziehungen und komplexe Objekte

Wie in Abbildung 1 graphisch dargestellt und im Kapitel `Object model` der ODMG beschrieben, können Objekte in Beziehung zu anderen Objekten stehen. Mittels Generalisierung wird die Struktur eines Objekteschemas mit dem Schema eines anderen Objektes erweitert. Es übernimmt dessen Merkmale wie Attribute und Methoden. Das Objekt einer Klasse B, die von Klasse A erbt, ist von Typ B als auch von Typ A. Der db4o-Anwender muss sich in der Regel keine Gedanken beim Speichern von erbenden Objekten machen. Die db4o-Engine erkennt die Struktur und hinterlegt sie automatisch in der Datenbank. Beim Zugriff auf Supertypen mit der Query By Example-Methode ist darauf zu achten, dass Example-Objekte unerwünschtes Verhalten erzeugen. Da sei auf die Methode `class()` verwiesen, die zu jeder Java-Klasse oder -Interface ein Klassenobjekt zurückliefert, das auch als Example übergeben werden kann, wie bei den SODA-Queries. Die andere Form von Beziehung ist die Referenzierung. Sie führt dazu, dass der Zustand eines Objektes u.a. die Summe aus dem Zustand anderer Objekte ist. Man spricht dann auch von einem komplexen Objekt. Diese Referenzierungen werden in der Implementierung gelöst durch Attribute, die nur ein Element oder durch Kollektionen, die mehrere Elemente aufnehmen können. Die Elemente können wiederum komplexe Objekte sein. Dadurch entsteht eine Strukturtiefe, die theoretisch beliebig ist. In die Tiefe zu gehen, ist eine Aufgabe, die ein Datenbanksystem zu bewältigen hat. In diesem Fall muss der db4o-Anwender etwas Vorarbeit leisten.

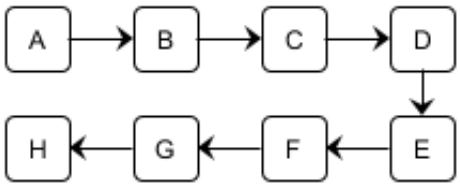

Figure 2: Beispiel einer Objektbeziehungsstruktur

3.5.1 Speichern und Updaten komplexer Objekte

Die Methode `store()` der ObjektContainer-Klasse löst bei Ausführung folgenden Mechanismus aus:

1. Prüfe, ob Objekt in Datenbank vorhanden ist.
2. Falls nicht, speichere Objekt in Datenbank.
3. Andernfalls update Objekt in Datenbank. Die alten Werte primitiver Datentypen wie `int`, `String` etc. werden mit den aktuellen Werten überschrieben.
4. Prüfe jedes einzelne, referenzierte Objekt ob es in Datenbank vorhanden ist.
5. Falls nicht, speichere referenziertes Objekt in Datenbank.
6. Andernfalls ignoriere Objekt.

Die Schritte 1 bis 5 sind intuitiv klar. Schritt 6 ist der Punkt, an dem der db4o-Anwender aufpassen muss. Werden bei einem Objektzugriff die Zustände referenzierter Objekte verändert, werden diese Änderungen standardmäßig bei der Objektspeicherung nicht übernommen. Ist aber ein sogenanntes kaskadierendes Updaten erwünscht, muss dies schon beim Öffnen der Datenbank in der Konfiguration angegeben werden. Dies zeigt folgender Codeausschnitt:

```
EmbeddedConfiguration config =
  Db4oEmbedded.newConfiguration();
config.common().objectClass(A.class)
  .cascadeOnUpdate(true);
ObjectContainer db = Db4oEmbedded
  .openFile(config, "datenbank.yap");
```

3.5.2 Laden tief strukturierter Objekte

In Abbildung 2 besteht eine verkettete Beziehung zwischen Objekt A und H. Es liegt eine Tiefe von sieben vor. Beim Laden und der Instanziierung von Objekt A aus der Datenbank ist die Aktivierungstiefe standardmäßig auf 5 eingestellt. Die referenzierten Objekte von F würden nicht automatisch mitgeladen und die entsprechenden Attribute auf Null gesetzt. Mit folgender Konfgurationseinstellung vor dem Öffnen der Datenbank wird die Aktivierungstiefe auf unbegrenzt eingestellt.

```
EmbeddedConfiguration config =
  Db4oEmbedded.newConfiguration();
config.common().objectClass(A.class)
  .cascadeOnActivate(true);
ObjectContainer db = Db4oEmbedded
  .openFile(config, "datenbank.yap");
```

Db4o stellt eine Vielzahl an Konfigurationsmöglichkeiten rund um die Aktivierungstiefe zur Verfügung. Zu finden sind die Methoden im Objekt `com.db4o.config.Configuration` und den assoziierten Klassen `ObjectClass` und `ObjectField`.

3.6 Transaktion

Ein Teil des Transaktions-Konzeptes schwang schon implizit in den Codebeispielen zur Öffnung und Schließen einer Datenbank mit. Eine Transaktion ist ein Aktionszeitraum, der mit dem Öffnen der Datenbank beginnt und mit dem Schließen der Datenbank endet. Datenbankvorgänge innerhalb dieses Zeitraums können mit dem Befehl `db.rollback()` rückgängig gemacht werden. Die Methode `db.commit()` speichert die letzten Vorgänge in die Datenbank. Das Schließen der Datenbank führt einen `commit` implizit aus. Bei Anwendung von `rollback` springt der Zustand demzufolge entweder auf den Zustand beim Öffnen der Datenbank oder zum letzten `commit`-Zeitpunkt innerhalb der Transaktion. Der Befehl `db.ext().refresh()` setzt ein instanziiertes Objekt auf den Zustand, wie er ihn in der Datenbank aufweist. Dazu muss der genannten Methode jeweils das betreffende Objekt übergeben werden.
Das Kapitel Transaktionen im Buch [8] geht tiefer auf diesen Sachbereich ein und prüft db4o auf die Datenbankanforderung ACID. Diese Abkürzung steht für Atomicity, Consistency, Isolation und Durability. In dem erwähnten Kapitel wird auch das Semaphoren-Konzept vorgestellt und wie db4o die speziellen Anforderungen paralleler Zugriffe regelt.

3.7 Schema Evolution

Schema Evolution ist in der Softwareentwicklung ein wichtiges Thema. Ein Programm ist selten endgültig fertig. Wartungsarbeiten und Featureerweiterungen bedingen fortlaufende Eingriffe in die Implementierung. Bei diesen Eingriffen kommt es mitunter zu Änderungen in dem Datenmodell und

in den Objektstrukturen. Db4o geht unterschiedlich mit den Refactoring-Vorgängen um.
Das Hinzufügen und Entfernen von Attributen behandelt die Engine automatisch. Werden Interfaces an Klassen angehängt, ist auch kein weiterer Aufwand nötig. Namensänderungen einer Klasse oder eines Feldes müssen so in der Datenbank angemeldet werden:

- Bezüglich Klasse:

```
configuration.common().objectClass(
  "com.db4odoc.strategies.refactoring
    .PersonOld").rename("com.db4odoc
      .strategies.refactoring.PersonNew");
```

- Bezüglich Feld

```
configuration.common().objectClass(
  "com.db4odoc.strategies.refactoring
    .PersonOld").objectField("name")
      .rename("vorname");
```

Dies soll als Beispiel reichen, um zu zeigen, dass Schema Evolution ihren Platz auch in db4o findet. Während bei Namensänderungen der Aufwand relativ gering ist, ist der Handlungsbedarf bei Typänderungen oder sogar Hierarchieänderungen ungleich größer, aber im Rahmen.[3]

3.8 Abgrenzung zu Versant Object Database

Der Blick auf die Homepage von Versant unterstreicht noch einmal den Open-Source und no-Business Charakter von db4o. Die objektorientierte Datenbank Versant Object Database ist spezialisiert auf Client/Server Architekturen und optimiert auf hoch skalierende Datenverwaltungsanforderungen[9]. Das Datenbanksystem unterstützt vorrangig Java, C# und C++. Für Abfragen stehen verschiedene Konzepte zur Verfügung wie die Versant Query Language oder die auf SQL basierende Object Query Language. Zur Demonstration folgt ein einfaches Codebeispiel einer Objektspeicherung und -abfrage[10]. Das Objekt pm steht für eine Instanz der Klasse PersistenceManager:

```
pm.currentTransaction().begin();
pm.makePersistent(person);
//schliesse Transaktion
pm.currentTransaction().commit();
Query q = pm.newQuery(Person.class);
q.setFilter("name == 'Matthias'");
Collection result = query.execute();
q.closeAll();
```

4. FAZIT

Mit db4o lässt sich leicht ein Datenbanksystem in eine Software integrieren. Es ist nicht nötig, eine neue Sprache zu lernen oder ein aufwendiges Framework zu installieren. Die Übertragung des Datenmodells in ein externes Schema fällt gänzlich weg. Der Abfragecode geht intuitiv von der Hand und lässt sich sehr gut von der übrigen Logik kapseln. Die einzubindende Library ist mit unter 1 MB sehr schlank und bei Abfragen von tiefstrukturierten Objekten in der Embedded Variante glänzt die Engine in der Performance[11]. Das sind gute Argumente für db4o, doch der fehlende Businesssupport des Anbieters schließt den professionellen Einsatz aus.

[3]Kapitel: Refactoring and Schema Evolution[2]

Ich persönlich würde db4o gerade wegen seiner Schlichtheit und seinem Open-Source-Charakter für Software-Prototypen auswählen, bei denen die Wahl der Datenbanktechnologie im Hinblick auf das Endprodukt noch keine Rolle spielt. Allgemein ist diese Arbeit aber auch zu verstehen als eine Einführung in ein alternatives NoSql-Datenbanksystem. Die hundertprozentige Objektorientierung einer Datenbanktechnologie macht Sinn in Kombination mit einer objektorientierten Programmiersprache hinsichtlich Konsistenz (Objekt zu Objekt), Implementierung (gleiche Sprache) und Performance (schnelle Abfrageergebnisse bei großen Datenmengen). Die BigData-Evolution wird die Entwicklung der objektorientierten Datenbanksysteme noch weiter voran treiben.

5. REFERENCES

[1] Douglas K. Barry R.G.G. Cattell and Contributors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers, San Francisco, California USA, 1999.

[2] Versant Corporation. *db4o-reference-java*. Versant Corporation, 255 Shoreline Drive, Suite 450 Redwood City, CA 94065 USA, unbekannt.

[3] wikinvest.com. Vsnt-topics-db40. Website, 2009. Online erhaeltlich unter `http://www.wikinvest.com/stock/Versant_(VSNT)/Db4o`; abgerufen am 19. Dezember 2015.

[4] Tim Schuermann. Objektdatenbank db4o fuer java und mono. *Linux-Magazin*, (9), 2007. Online einsehbar unter `http://www.linux-magazin.de/Ausgaben/2007/09/Konservierungsmittel`; abgerufen am 19. Dezember 2015.

[5] Versant Corporation. *db4o-8.0-tutorial*. Versant Corporation, 255 Shoreline Drive, Suite 450, Redwood City, CA 94065 USA, Jahr unbekannt. in db4o 8.0 Distribution enthalten.

[6] Carl Rosenberger William R. Cook. *Native Queries for Persistent Objects*. `http://www.cs.utexas.edu/users/wcook/papers/NativeQueries/NativeQueries8-23-05.pdf`, 2006. Online abgerufen am 19. Dezember 2015.

[7] Siddhartha Rai William R. Cook. *Safe Query Objects*. `http://www.cs.utexas.edu/users/wcook/papers/SafeQuery05/SafeQueryFinal.pdf`, unbekannt. Online abgerufen am 19. Dezember 2015.

[8] Ina Brenner. *Datenbankentwicklung mit db4o*. Books on Demand GmbH, Norderstedt, 2007.

[9] Actian Corporation. Vod product description. Website. Online einsehbar unter `http://esd.actian.com`; abgerufen am 19. Dezember 2015.'... significant large-scale data management requirements.'.

[10] Versant Corporation. *Versant JDO Programmer's Guide*. 500 Arguello Street, Suite 200 Redwood City, CA 94063 USA, 2015. Online erhaeltlich unter `http://esd.actian.com`; abgerufen am 19. Dezember 2015. Kapitel 2-5.

[11] polepos.org. Poleposition open source database benchmark. Website. Online erhaeltlich unter `http://polepos.sourceforge.net/results/PolePositionEmbedded.pdf`; abgerufen am 19. Dezember 2015.