Zusammenfassung des Vortrags

“Persistenz von Objekten”

Sven Schliesing, 13. April 2005

sven.schliesing@informatik.haw-hamburg.de

Viele Teilprojekte im Projekt “Ferienclub” des Masterstudiengangs “Informatik” benötigen eine einheitliche Datenbasis. Es werden abrechnungs-, verwaltungs- und gesellschaftsrelevante Daten anfallen die eine zuverlässige, simple aber ebenso mächtige Datenbasis/-verwaltung benötigen. Die Schnittstellen sollen einfach anzubinden, aber ebenfalls so ausgereift sein, dass eine Analyse z.B. mittels Data Warehouse / Business Intelligence möglich ist.

Inhalt

Einleitung und Motivation 2
Schnittstellen und Backend-Struktur 3
Datenbanktypen 4
relational 4
objekt-relational 4
Objektdatenbank 4
Objektdatenbank: DB4Objects 5
Resourcensparend: 5
hohe Performance: 5
zuverlässig: 5
einfach zu implementieren: 5
portierbar: 5
Objektdatenbank: ObjectDB (als JDO-Implementation) 6
“The big picture” und “Offene Fragen” 8
Quellen 9

Einleitung und Motivation

Persistenz bezeichnet "*die Fähigkeit [...], Datenstrukturen (oder Objekte) in nicht-flüchtigen Speichermedien wie Dateisystemen oder Datenbanken zu speichern.*" (Freenet Lexikon, http://lexikon.freenet.de/Persistenz_%28Informatik%29).

Da es sich bei den zu speichernden Daten weniger um grosse Mengen an Binärdaten als vielmehr um worl strukturierte Daten aus den zugreifenden Teilprojekten handelt fiel die wahl der Persistenzhaltenden Architektur auf eine Datenbank. Genauer gesagt auf eine "Objektdatenbank".

Gründe hierfür sind die einfache Einbindung einer Objektdatenbank sowie der einfache Übergang von Daten (meist in Form von Objekten) in die Persistenzschicht. Einem Dateisystem mangelt es an passender Struktur um die Datenhaltung sowie Datenabfrage performant und damit sinnvoll zu gestalten. (Folie 3) Die weitere Möglichkeit eine relationale Datenbank zu verwenden wird später genauer betrachtet.

Die Dienste, die die Persistenzschicht anbieten soll ist neben den klassischen Funktionen wie Speichern und Laden von Objekten/Daten ebenfalls eine Rechtespeicherung sowie deren Verwaltung. (Folie 4)

Desweiteren wird die Abspeicherung eines sogenannten "Abstract" unterstützt. "Abstract" meint in diesem Fall eine "Vorschau" auf das eigentliche Objekt. Ein klassischer Anwendungsfall wäre hier ein "Thumbnail" eines Bildes oder ein Auszug aus einem Video / einem Musikstück.

```
module PersistenceModule {
	interface Persistence {
		struct genericObject {
			long owner;
			long time;
			octet[] data;
		};

		typedef octet[160] OpaqueKey;
		exception InvalidKeyException{};

		OpaqueKey saveObject(in genericObject obj);
		genericObject loadObject(in OpaqueKey key, out genericObject obj)
			raises InvalidKeyException;
		genericObject loadAbstract(in OpaqueKey key, out genericObject obj)
			raises InvalidKeyException;
};
```

Weitere denkbare Methoden wären u.A.:

- loadObjectByAlias(ein/aus objectAlias : string(idl))
- loadAbstractByAlias(ein/aus objectAlias : string(idl))
- getOwnerByObject(ein/aus objectId : long(idl))

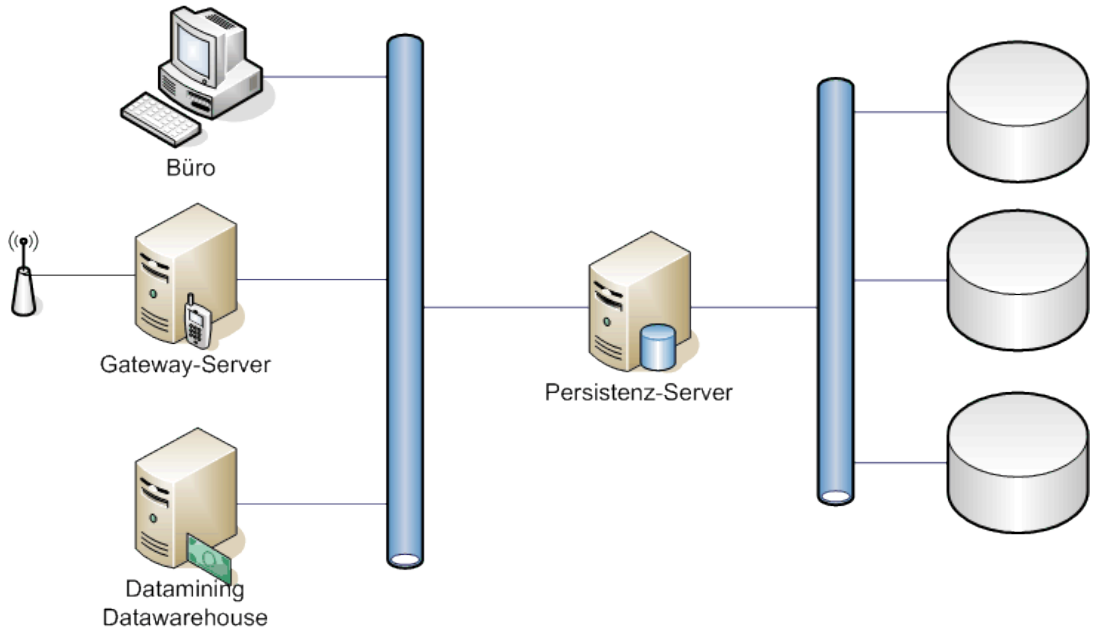
Schnittstellen und Backend-Struktur

Um die einfache Anbindung anderer Anwendung/Projekte an die Persistenz zu gewähren, wird eine vielzahl von Schnittstellen bereitgestellt. Zum Zeitpunkt des Vortrags wurde vorerst eine einfache, generische Schnittstelle beschrieben.(Folie 5 ff.) Im Laufe des Projekts werden anwendungsspezifische Schnittstellen hinzugefügt und angepasst.

Die Schnittstellen werden, wie dieses Schaubild (Folie 9) zeigt durch einen Persistenz-Server bereitgestellt. Dieser transparente Zugriff auf die Datenhaltung bietet die Möglichkeit, mehrere Datenbankserver redundant zu betreiben sowie die Datenhaltung, wenn gewünscht, heterogen (objektbasiert und relational) zu gestalten.

Auch wäre eine gesonderte Abspeicherung von, möglicherweise doch anfallenden, Binärdaten wie Musikstücken, Bildern oder Videos in einem Dateisystem möglich, ohne dass die, die Persistenzschicht nutzende, Anwendung dieses Detail kennen muss.

Ebenso besteht die Möglichkeit einen Gateway-Server bereitzustellen, der mit der mobilen Persistenzschicht von “Lutz Behnke” kooperiert.



Datenbanktypen

Bevor die Wahl für dieses Projekt auf die Verwendung einer Objektdatenbank gefallen ist, wurden die Möglichkeiten die andere Modelle bieten abgewogen. Als Alternative wurde hier die “relationale” sowie “objekt-relationale” Datenhaltung betrachtet.

relational

Die Verwendung einer relationalen Datenbank zur Speicherung von Objekten als Datencontainer bringt das Problem des “Impedance mismatch” mit sich.

Das hier bestehende Problem zeichnet sich dadurch aus, dass es beim Übergang von Objekten in eine relationale Datenhaltung zu einem “Architekturbruch” kommt. Die Konzepte der objektbasierten Programmierung basieren auf der Software-Architektur in der sie verwendet werden. Die Konzepte der relationalen Datenhaltung wiederum basieren auf Konzepten aus der Mathematik. Beim Übergang zwischen diesen beiden Welten kommt es unweigerlich zum Problem des Übergangs:

Objekte müssen in einen relationalen Aufbau “eingepasst” werden: Ein Mapping ist notwendig.

Ebenfalls besteht ein Problem wenn man unbekannte Objekte “persisten” möchte. Die Struktur ist unbekannt und macht ein (nicht generisches) Mapping nur mit immensen Aufwand möglich. Die Einpassung von Objekten mit unbekannter Struktur in generische relationale Modelle erschwert es jedoch wiederum Operationen (z.B. suchen) performant auf ihnen auszuführen.

objekt-relational

Objekt-relationale Datenbanken verstehen sich als Bindeglied, als Hybrid, zwischen den Welten der Objekt- und relationalen Datenbanken. Sie sind um obekt-Funktionen erweiterte relationale Modelle.

Objektdatenbank

Die Möglichkeiten die uns eine native Objektdatenbank in unserem Projekt bietet sind besonders wertvoll. Es ist nicht nur eine einfache Integration möglich, Objektdatenbanken bieten auch den kürzesten Weg zwischen “lebenden” Objekten und der Datenhaltung. Der “Impedance mismatch” wie er bei relationalen Datenbanken besteht tritt hier nicht auf. Außerdem ist es möglich selbst auf generischen Daten strukturierte Abfragen auszuführen, sodass Funktionen wie eine Suche problemlos implementiert werden kann. Zusätzlich ist es möglich eine Klassenhierarchie zu implementieren, die eine Abfrage von Daten für Analyse-Tools (Data-Warehouse / Business Intelligence) erleichtert.

Lediglich das generelle Defizit an ausgereiften Produkten sowie Know-How können eine Einbindung erschweren. Dies wird jedoch nicht als Hindernis sondern als Herausforderung gesehen.

Objektdatenbank: DB4Objects

Als Objektdatenbank wurde primär “DB4Objects” betrachtet. Laut Herstellerangaben:

- resourcensparend
- hohe Performance
- zuverlässig
- einfach zu implementieren
- portierbar

In wie weit diese Aussagen zutreffen konnte schon grob in den Vorbereitungen zu diesem Vortrag getestet werden.

Resourcensparend:

Die Bibliothek der Datenbank, die bequem in eine Anwendung integriert werden kann (es wird nicht zwingend ein extra Server benötigt), ist lediglich 250kB gross und fällt somit unter der, nicht gerade speicherschonenden, JVM nicht sonderlich ins Gewicht.

hohe Performance:

Die Performance konnte, aufgrund von fehlenden Datenmengen, nicht ausreichend getestet werden und so ist dieser Punkt fraglich. Der auf Folie 25 betrachtete Benchmark ist mit Vorsicht zu geniessen da Performance-Vorteilen von teilweise dem 44x gegenüber einer Mapping-Lösung auf eine relationale Datenbank (Hibernate) doch sehr fraglich sind.

zuverlässig:

DB4Objects hält sich, wie alle anderen ernstzunehmenden Datenbanken auch, an das ACID-Prinzip (Atomicity, Consistency, Isolation, Durability). In wie weit dieses gegeben und auch zuverlässig ist, wird ebenfalls im Laufe des Projekts weiter untersucht.

einfach zu implementieren:

Die Einbindung der Bibliothek in ein bestehendes oder neues Programm läuft, wie der Hersteller, verspricht problemlos und unkompliziert ab. In wenigen Zeilen ist eine Datendatei ausgewählt und das Interface bereit.

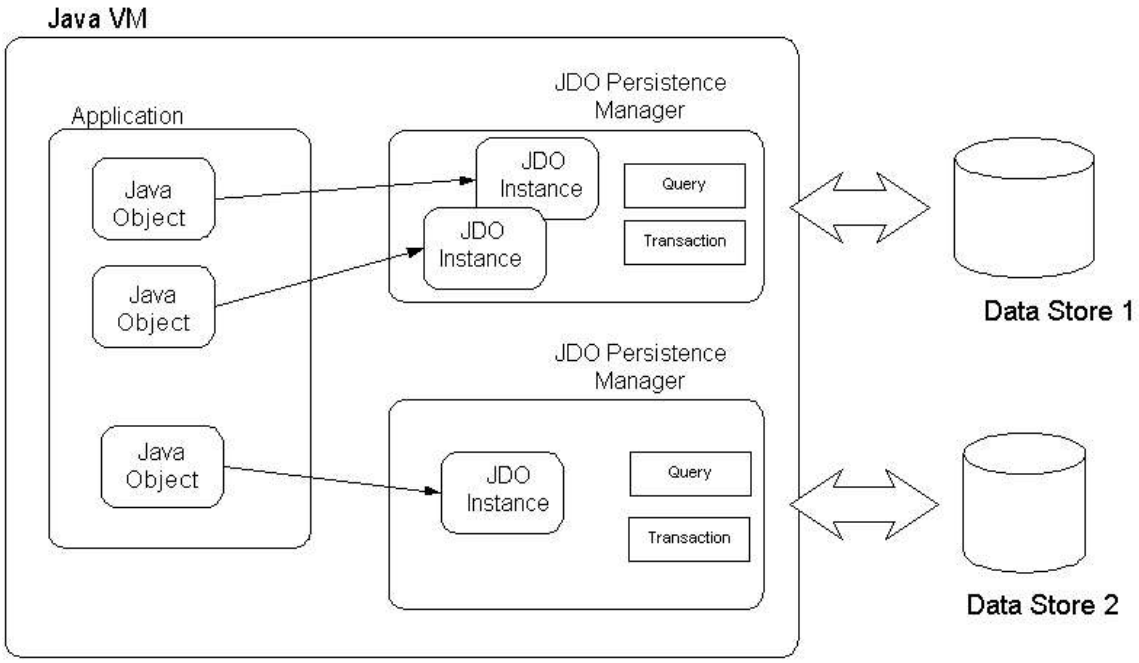
portierbar:

Da DB4Objects komplett in Java geschrieben ist, läuft diese Datenbank auf allen System für die eine JVM (J2SE) existiert. Ob sie auch auf kleineren Java-Versionen wie der J2ME läuft ist nicht bekannt.

Objektdatenbank: ObjectDB (als JDO-Implementation)

JDO ist eine von Sun entwickelte API. Sie soll dem Programmierer eine einheitliche Persistenzschicht, unabhängig von der verwendeten Datenbank, bieten.

Als Implementation von JDO wurde für diesen Vortrag “ObjectDB” gewählt.

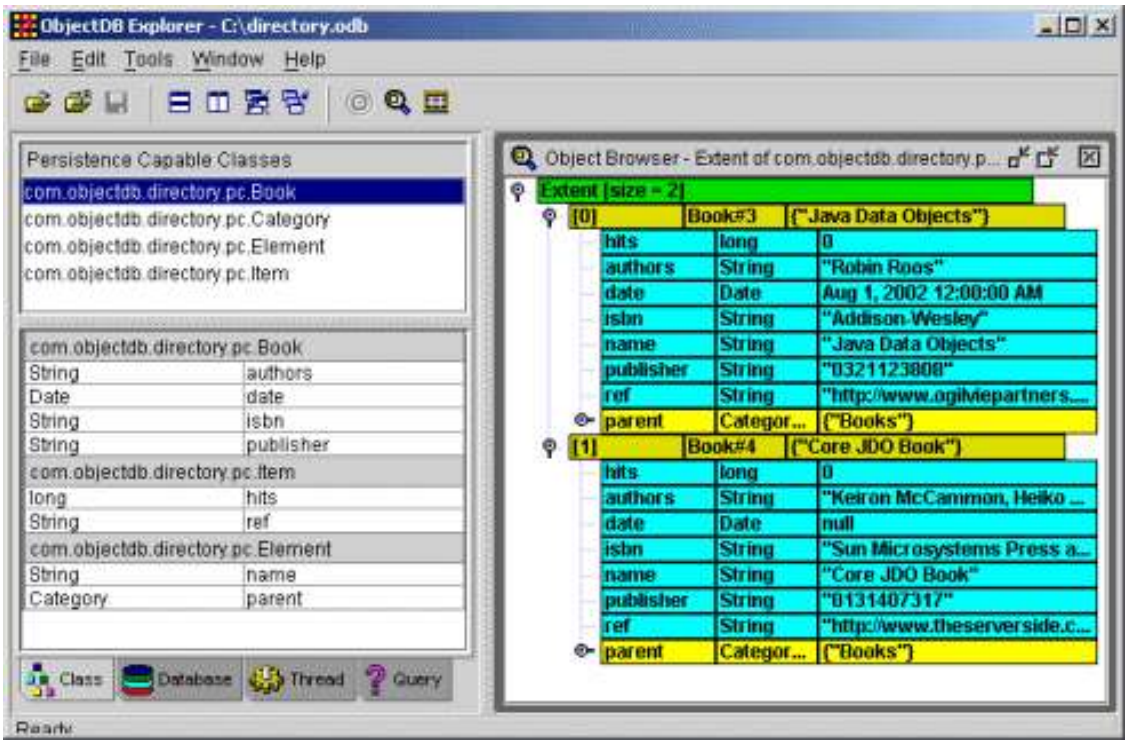


Quelle: http://www.javaworld.com/javaworld/jw-04-2002/images/jw-0412-jdo1.gif

Ebenso wie DB4Objects ist es bei ObjectDB möglich, die Datenbank “embedded” (im eigentlichen Programm) oder im “Client-Server-Mode” zu betreiben.

Eine Einbindung und Datenabfrage ist ebenso unproblematisch möglich.

Zusätzlich wird ein sogenannter “ObjectDB Explorer” angeboten, der eine anschauliche Darstellung der in der Datenbank gespeicherten Objekte bietet:

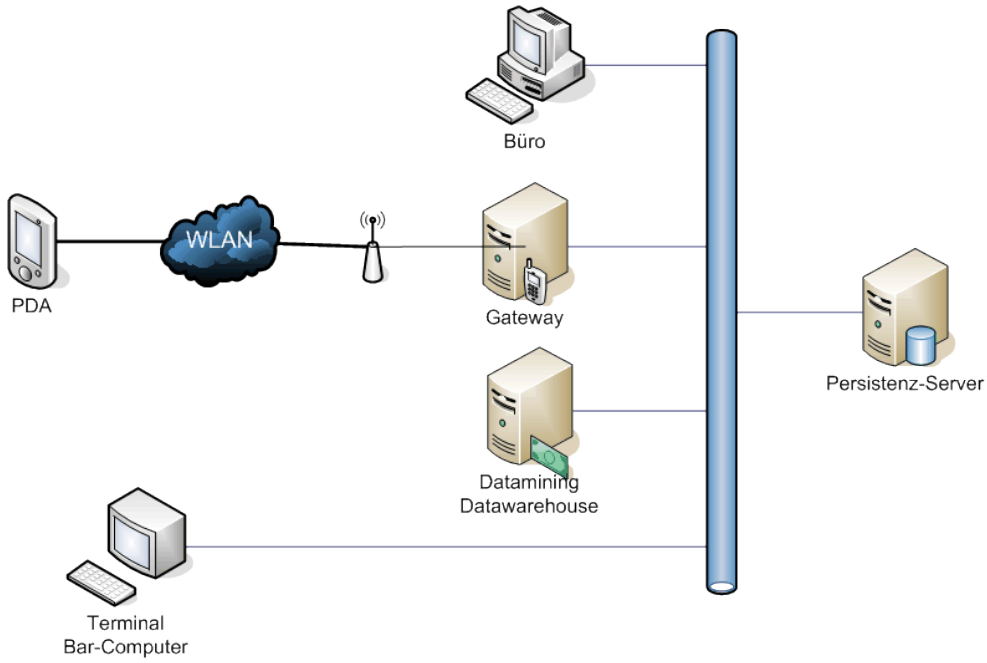


Quelle: http://www.objectdb.com/database/jdo/java-explorer/

“The big picture” und “Offene Fragen”

Im laufe dieses Vortrags ist klar geworden, wo der angebotene Persistenz-Dienst angesiedelt ist.

Er stellt eine sichere Datenhaltung zur Verfügung, bietet ausreichend Schnittstellen für andere Teilprojekte sowie für Analyse-Tools.



Offen sind jedoch ein paar Punkte, die im Verlaufe des Projekts im Wintersemester 05/06 geklärt werden müssen.

Welche weiteren Schnittstellen werden benötigt? Ist vielleicht eine ausgefeilte generische ausreichend für alle Anwendungsfälle?

Wie ist die Performance? Im Speziellen bei großen Datenmengen (Video, Bilder, Musik, etc)? Muss eventuell eine Hybrid-Lösung aus Objektdatenbank + Dateisystem implementiert werden?

Ist das Konzept der Objektdatenbanken überhaupt ausgereift genug um in diesem Projekt eingesetzt zu werden?

Quellen

- Object Database vs. Object-Relational Databases, August 1997: http://www.ca.com/products/jasmine/analyst/idc/14821E.htm
- The Object-Oriented Database System Manifesto, 1995: http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html
- The Database Behind the Brains, unbekanntes Datum http://www.db4objects.com/about/productinformation/whitepapers/db4o%20Whitepaper%20-%20The%20Database%20Behind%20the%20Brains.pdf
- Relationale Datenbank: http://de.wikipedia.org/wiki/Relationale_Datenbank
- The Object-Relational Impedance Mismatch: http://www.agiledata.org/essays/impedanceMismatch.html
- EJB/JDO Persistence FAQ: http://java.sun.com/j2ee/persistence/faq.html