Object Oriented Persistence Middleware

System Architectures
Principles and techniques
Persistence Managers:
OBJ, Hibernate

References

- S. Ambler: The Design of a Robust Persistence Layer
  http://www.ambysoft/persistenceLayer.pdf
- S. Ambler: Encapsulating Database Access
  http://www.agiledata.org/essays/implementationStrategies.html
- Java Data Objects
  http://access1.sun.com/jdo/
- The Object Relational Bridge (OBJ)
  http://db.apache.org/ojb/index.html
- Relational Persistence For Idiomatic Java
  http://www.hibernate.org/
- Gopalan Suresh Raj: Java Data Objects (JDO)
  http://my.execpc.com/~gopalan/java/jdo/jdo.html
  http://citeseer.nj.nec.com/yoder98connecting.html
Architecture for Persistent objects

- **Design principles**
  - Keep language clean from DB access code
  - "persistence manager" between client code and DB
  - DB specific code generated for use by the persistent mgr

- **API for Application Program?**
  - "Business Logic"

- **Mapping to Database?**
  - ODMG QQL
  - EJB-QL
  - Java Data Objects JDO
  - Ad hoc

- **Persistence Layer**

  - **Benefits**
    - Reduction of database and object schema coupling
    - Implementation of DB related code in one place: less effort for adapting to DB schema changes ("evolution")
    - Makes application development easier: concentrate on application logic but DB access code
    - Allows for data oriented business rules
      - e.g. "if an employee earns more than 100K he/she does not get overtime payment"
    - But integrity constraints?

  - **Risks**
    - Direct DB access is easier for small applications
    - investment required
    - Loose of control over DB access, e.g. when to write objects to DB
      - example: EJB Container Managed Persistence
    - Replication of DB functionality
    - Programming language dependent
Principle Persistence Architectures

- Traditional: Direct database access
  - Business objects

- Separate Data Access objects

- Persistence Layer
  - Transaction facilities
  - Query language mapping
  - ... and much more

- Services

Direct DB acceso

- Customer

- Violates most of the design principles
  - No "separation of concerns"
  - ... which means here: application logic and DB access should be separated
    - Schema evolution
    - DB-vendor independence (but language independence is missing from most frameworks)
Direct data access

- SQL / OQL/.. statements for implementing the "customer" operations
- JDBC standard when using Java
- ... but code is written quickly
- viable for small applications

Data Access Objects (DAO)

- Encapsulation of DB access logic in classes independent from business objects
- Typically one access object for each business object
- Access Objects may or may not follow standards (ActiveX ADO, JDO),
Persistence Framework

- Mapping between DB data and objects defined in a Meta Data Repository (XML document, database, …)
- Basic functionality: create, read, update, write, transaction support
- Enhanced: fault tolerance, mapping generation, caching etc.

Features

- Implicit persistence:
  - framework automatically makes business objects persistent
  - no action taken by application program
  - Examples: Enterprise Java Beans, Java Data Objects (container managed)
- Explicit persistence
  - application program indicates when to save
  - Most common in Persistence frameworks
- DB access generated:
  - dynamically: flexible
  - at compile time: simpler, better performance
  - during system start up
  (example: Hibernate framework)
Persistence Framework

- Reading a business object via persistence framework

Customer
Persistence Framework
Mapping class

read(customer)
getValues()

keyvalues

getMap

Construct select

Result set

Set data values

Persistence Framework

- Steps during Data access in Meta Data driven

Query Processor

Database(s)

Business Object

1. Meta Data Query
2. Meta Data Query
3. Select Statement
4. Select Statement
5. Database Result
6. Query Result Representation
7. Query Result Representation
8. Query Result Representation

Figure © S. Amber
Meta Data driven persistence architecture

Example(1)  Meta data Query for
"Return all orders placed Jan. 27, 2003"

<Query>
  <Search For>Order</Search For>
  <Clause>
    <Attribute>"dateOrdered"</Attribute>
    <Comparison>"="</Comparison>
    <Value>"27-Jan-2003"</Value>
  </Clause>
  <Clause>
    <Attribute>"subtotalBeforeTax"</Attribute>
    <Comparison>"\\geq"</Comparison>
    <Value>"1000.00"</Value>
  </Clause>
</Query>

Example(2): Mapping for Order class / table

<table>
<thead>
<tr>
<th>Property</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order.orderID</td>
<td>Order.OrderID</td>
</tr>
<tr>
<td>Order.dateOrdered</td>
<td>Order.DateOrdered</td>
</tr>
<tr>
<td>Order.dateFulfilled</td>
<td>Order.DateFulfilled</td>
</tr>
<tr>
<td>Order.getTotalTax()</td>
<td>Order.Tax</td>
</tr>
<tr>
<td>Order.subtotalBeforeTax</td>
<td>Order.SubtotalBeforeTax</td>
</tr>
<tr>
<td>Order.shipTo.personID</td>
<td>Order.ShipToContactID</td>
</tr>
<tr>
<td>Order.billTo.personID</td>
<td>Order.BillToContactID</td>
</tr>
<tr>
<td>Order.lastUpdate</td>
<td>Order.LastUpdate</td>
</tr>
</tbody>
</table>

Mapping typically very simple
Meta Data driven persistence architecture

- Example (3)

```sql
SELECT *
FROM Order
WHERE Order.DateOrdered = '2003-01-27'
AND Order.SubtotalBeforeTax >= 1000.00
```

- Result presentation generated as well

- More sophisticated mappings needed
  … but not provided in most frameworks

- How to employ efficient database query processing when loading objects?

Persistence framework

- Why complex mappings are needed

Customer object with two orders and two / one items

Data base representation using three tables

Wanted: All Customers located in Berlin with orders of july 30 with items not yet delivered
(Items may be delivered individually)
Persistence framework

- Complex mapping (cont.)

```sql
SELECT c.name, c.something, o.id, o.etc, i.no, i.etc
FROM customer c, order o, item i
WHERE o.cid = c.cid
    AND c.city = "Berlin"
    AND o.date = "7-30-2003"
    AND i.id IN (SELECT j.id
                  FROM item j
                  WHERE j.oid = o.oid
                  AND j.delivered = false)
```

- Map result tuples to objects: customer, order, item
- Inefficient alternative:
  - Access tables independently
    Many independent database accesses:
    customer, for each customer his order, for
    each order its items: expensive
  - Implementation issue: Lazy loading, otherwise the
    object construction will take some time….

Services

- Service:
  an operation offered by a computing entity that can
  be invoked by other computing entities (Ambler)

- Examples
  - Corba
  - Web services
  - Stored procedures
Goal: Encapsulate access to legacy data

Customer

Read
Customer
Service

Wrapper
class

GetDataValues()

read(id)

readCustomer(id)

Construct select

select

Result set

Key values

XML

Set data values

Customer data

XML Build

Comparison: DB access from business object

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very simple approach.</td>
<td>Directly couples your object schema to your data schema.</td>
<td>At beginning of a project when your persistence approach is still in flux.</td>
</tr>
<tr>
<td>Can develop code very quickly.</td>
<td>Application developers need to learn database access language (e.g. SQL)</td>
<td>For small applications (less than 20 business classes) and/or prototypes.</td>
</tr>
<tr>
<td>Can support access to very bad data designs (although performance may suffer).</td>
<td>Database refactoring impeded due to high coupling.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Difficult to reuse database access code.</td>
<td></td>
</tr>
</tbody>
</table>

© S. Amber
### Comparison: DB access via Data objects

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database access code encapsulated into its own set of classes.</td>
<td>Object schema still coupled to your data schema, via the data access objects.</td>
<td>Medium-sized application (20-100 business classes).</td>
</tr>
<tr>
<td>Business classes no longer coupled to database.</td>
<td>Application developers need to learn SQL.</td>
<td></td>
</tr>
<tr>
<td>Database refactoring easier due to lowered coupling.</td>
<td>Often platform specific.</td>
<td></td>
</tr>
<tr>
<td>Can support access to very bad data designs (although performance may suffer).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Possible to reuse data access objects.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Comparison: DB access via persistence framework

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application programmers do not need to know the data schema.</td>
<td>Perceived performance impact to your applications (if the framework is poorly built).</td>
<td>Medium and large sized applications. When it is common practice within your organization to use a persistence framework.</td>
</tr>
<tr>
<td>Application programmers don’t even need to know where the data is stored.</td>
<td>Requires reasonably clean data designs because the framework may not support the overly complex mappings.</td>
<td></td>
</tr>
<tr>
<td>Frameworks reflect performance expertise of its builders.</td>
<td>Often platform specific.</td>
<td></td>
</tr>
<tr>
<td>Administration facility can ease database refactoring because it simplifies impact analysis by tracing columns to object attributes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Administration facility aids performance tuning because it makes it easy to change mappings.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Possible to reuse framework and mapping meta data between applications.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Comparison: Service access to data

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential to create platform independent services.</td>
<td>Web services standards and tools still evolving.</td>
<td>Medium to large sized applications. Whenever an appropriate service already exists that you can reuse.</td>
</tr>
<tr>
<td>Web services quickly becoming an industry standard.</td>
<td>Performance becomes a problem when combining several services in serial or simply when services are invoked across a network.</td>
<td></td>
</tr>
<tr>
<td>Supports reuse between applications.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Persistence Frameworks

- **ObjectRelationalBridge (OBJ)**
  - O/R Mapping tool and persistence framework by Apache (open source)
  - Supports various interfaces to application objects
    - ODMG 3.0
    - JDO
    - CustomLanguage OBJ
  - Sophisticated mappings (1:n, n:n)
  - Cache
  - Lazy loading
  - Distributed lock management (pessimistic)
  - In addition optimistic synchronisation
  - Prefetching relations
  - Interface to many relational DBS
Persistence Frameworks: Hibernate

- Hibernate (free software, LGPL)
  - mapping at startup time
  - High performance
  - Mapping using reflection facilities
  - Custom language
  - ODMG 3.0 as beta implementation
  - Lazy loading
  - Cache and query cache (optional)
  - Outer join, loading object graph with one select
  - Optimistic synchronisation
  - Extensive documentation

Hibernate example

- The database

```sql
CREATE TABLE `users`
    (`LogonID` varchar(20) NOT NULL default '0',
    `Name` varchar(40) default NULL,
    `Password` varchar(20) default NULL,
    `EmailAddress` varchar(40) default NULL,
    `LastLogon` datetime default NULL,
    PRIMARY KEY (`LogonID`));
```

From tutorial by Glen Smith
### The application classes

```java
public class User {
    private String userID;
    private String userName;
    private String password;
    private String emailAddress;
    private Date lastLogon;
    public String getID() { return userID; }
    public void setID(String newUserID) { userID = newUserID; }
    // ... a bunch of other properties // using getXXX() and setXXX()
    go in here...
}
```

### Defining the mapping

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping.dtd">
<hibernate-mapping>
    <class name="dbdemo.User" table="users">
        <id name="ID" column="LogonId" type="string">
            <generator class="assigned"/>
        </id>
        <property name="userName" column="Name" type="string"/>
        <property name="password" type="string"/>
        <property name="emailAddress" type="string"/>
        <property name="lastLogon" type="date"/>
    </class>
</hibernate-mapping>
```
Hibernate example

- Define Database (property file)
- Create Datastore object
  - Datastore ds = Hibernate.createDatastore();
  - ds.storeFile("MyMappingFile.hbm.xml")
- Build session object session
- Business logic, custom query language

```java
List myUsers = session.find("from user in class dbdemo.User where user.ID = ?", "joe_cool", Hibernate.STRING);
if (myUsers.size() > 0) {
  for (Iterator i = myUsers.iterator(); i.hasNext(); ) {
    User nextUser = (User) i.next();
    System.out.println("Resetting password for User: "+nextUser.getUserName());
    nextUser.setPassword("secret");
  }
} else {
  System.out.println("Didn't find any matching users..");
}
```
- Close session