Patterns for Making Business Objects Persistent in a Relational Database

Presented By:
Joseph W. Yoder
joeyoder@joeyoder.com
www.refactory.com

More Info

For additional information see http://www.joeyoder.com/Research/objectmappings

Collaborators

Ralph Johnson johnson@cs.uiuc.edu

Quince Wilson qwilson@issintl.com

The Refactory, Inc. www.refactory.com

Goals

- Look at patterns for making objects persistent in a non-object world
- Learn a framework for mapping your objects to a relational database
- Take some sample code or ideas back with you that you can use in your development environment

Overview

- Motivation & Problems
- Patterns for Mapping Objects to RDBMS's
- How we Developed our Framework
- The Design of our Framework
- The Relational Database Side of Things
- Meta-architecture for mapping objects to RDBMS's
- Summary

Motivation

Systems are often developed where mappings to a relational database for domain values are needed. Quite often relational calculus and the maturity of relational databases are exactly what one needs. Other times it might be that the corporate policy is to use a relational database rather than an object-oriented database.

Problems with OO-RDBMS Mappings

- Impedance Mismatch of technologies

Objects - hierarchies, types, composition,
polymorphism, relate code and data

Relations - rows, tables, relational calculus
permanent storage, data access

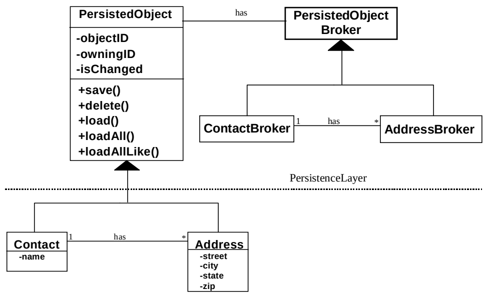
Mapping Objects To RDBMS Persistence Pattern Language

- Persistence Layer
- CRUD
- SQL Code
- Attribute Mapping Methods
- Type Conversion

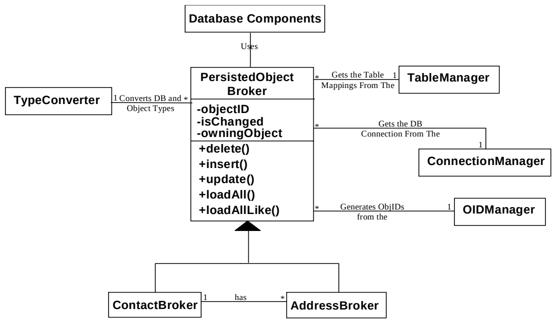
Mapping Objects To RDBMS Persistence Pattern Language

- Changed Manager
- OID Manager
- Transaction Manager
- Connection Manager

The Patterns in Action



The Patterns in Action



Persistence Layer

Problem:

How to make objects persistent to a non object-oriented storage application such as a relational database. This should be accomplished in such a fashion as to relieve the developers from having to know the exact implementation.

Persistence Layer

Solution:

Provide a Persistent Layer in which objects are able to populate themselves from a data storage source as well as save themselves back to the data storage source. This is really a special case of building a layer to protect you from changes. It is similar to *Adapters* and *Facades*. A standard interface is provided in which all objects that need to be persisted interface to.

Persistence Layer

Discussion:

- Use a Layered Object. Subclass each domain object from an abstract class PersistedObject.
- Provide a *Broker* that can read or write domain objects to or from the database.
- Compose each domain object from a set of data objects that have a one to one mapping to the database tables.

13

Persistence Layer

Using a Layered Object:

- Inherit behavior for persisting to database.
- Overwrite methods for reading and writing values to and from the database.
- Layer isolates developer from database details.
- Easy to Write SQL mapping
- Have to inherit from a PersistedObject

14

Persistence Layer

Using a Broker:

- Similar to a Layered Object in that you go through a separate layer to persist your objects.
- SQL Code is kept separate from your domain.
- Can inherit from any object and still have a persistent mapping.
- All SQL code is conglomerated together.

15

Persistence Layer

Using Data Objects:

- Create a one-to-one mapping between tables in the database and simple data objects.
- Create domain objects by using data objects.
- Simple and easy to map to…can easily map to any database.
- Let data objects know when they are dirty and save themselves when you commit.
- Have to handle complicated queries through views or multiple data objects

16

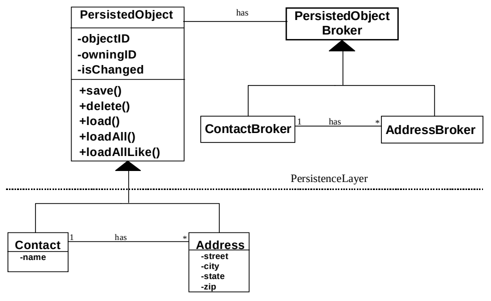
Persistence Layer with Broker

Our Solution:

We create a Persisted object that any object that we want to be persisted can inherit from. The details of how the persistence is done is hidden from the user. It also provides a place where mapping to new types of persistence storage can be created and integrated into the system without affecting the application code. This persisted object talks to a Broker object to separate the details of the SQL from the domain object.

17

Persisted Class Diagram



18

Persisted Object with Broker

- Abstract class
- Standard Interface to Persistence Layer
- Supports General CRUD Operations
  - (create, read, update, and delete)
- Broker Sub-Classes overwrite the specific mappings to the database

19

Persisted Object (Java Class)

```
public abstract class PersistedObject extends
   AbstractManager{
   private double objectId;
   private double ownerId;
   private Timestamp lastChanged;
public abstract PersistedObjectBroker broker();
```

20

Persisted Object (Java Class)

```
public void setLastChanged(Timestamp newLastChanged) {
   lastChanged = newLastChanged;}
public void setObjectId(double newObjectId) {
   objectId = newObjectId;}
public void setOwnerId(double newOwnerId) {
   ownerId = newOwnerId;}
public Timestamp getLastChanged() {
   return lastChanged;}
public double getObjectId() {
   return objectId;}
public double getOwnerId() {
   return ownerId;}
```

21

Persisted Object (basic operations)

```
public boolean delete()  throws SQLException {
   return
   broker().delete(broker().brokerFor(),((PersistedObject)this)));}
public boolean insert()  throws SQLException {
   return (broker().insert((PersistedObject)this));}
public boolean update()  throws SQLException {
   return (broker().update((PersistedObject)this));}
public boolean save() throws SQLException {
   if (getObjectId() == 0){
        return insert();
   } else {
        return update();};}
```

22

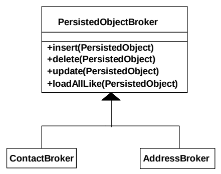
Persisted Object (reading objects)

```
public PersistedObject load(double anObjectId)
  throws SQLException  {
    if (anObjectId == 0){
       return this;};
    setObjectId(anObjectId);
    ArrayList al =
  (broker().loadAllLike((PersistedObject this));
    ListIterator aList = al.listIterator();
    PersistedObject anObject = null;
    while (aList.hasNext()) {
      anObject = (PersistedObject)aList.next();};
  return anObject;}
public ArrayList loadAllLike()  throws
  SQLException  { return
  (broker().loadAllLike((PersistedObject)this));}
```

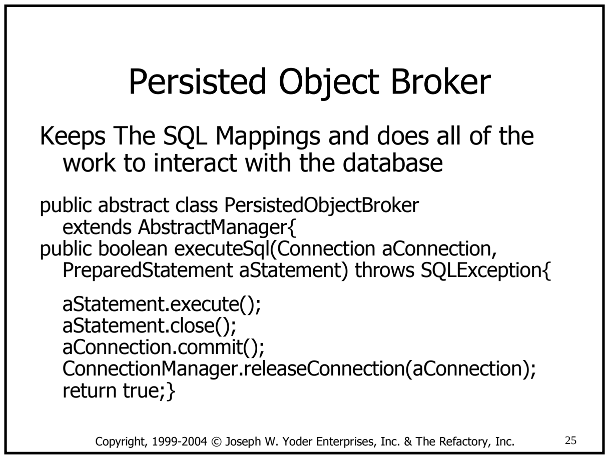
23

Persisted Object Broker

- Keeps The SQL Mappings and does all of the work to interact with the database
- Persistent Objects define a subclass of the Broker which defines the database mappings for that object



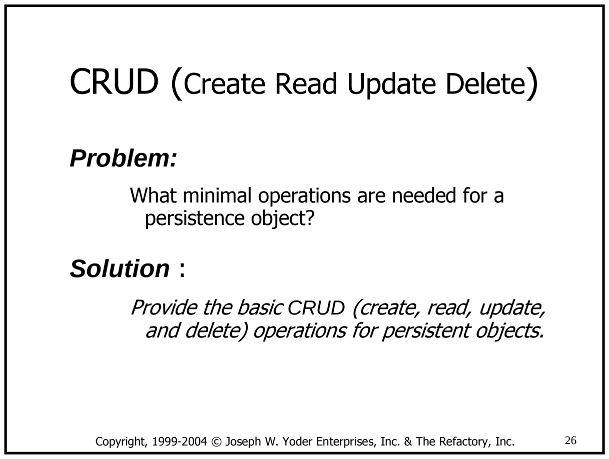
24



Persisted Object Broker

Keeps The SQL Mappings and does all of the work to interact with the database

```
public abstract class PersistedObjectBroker
   extends AbstractManager{
public boolean executeSql(Connection aConnection,
   PreparedStatement aStatement) throws SQLException{

   aStatement.execute();
   aStatement.close();
   aConnection.commit();
   ConnectionManager.releaseConnection(aConnection);
   return true;}
```



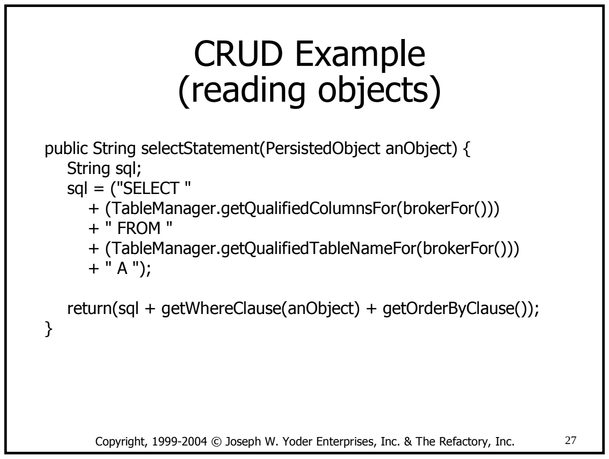
CRUD (Create Read Update Delete)

Problem:

What minimal operations are needed for a persistence object?

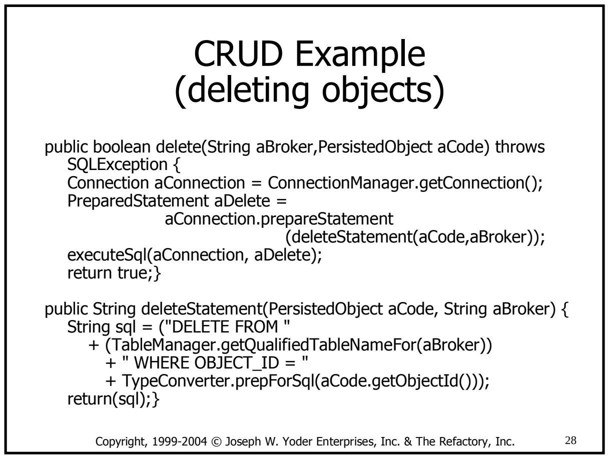
Solution :

Provide the basic CRUD *(create, read, update, and delete) operations for persistent objects.*



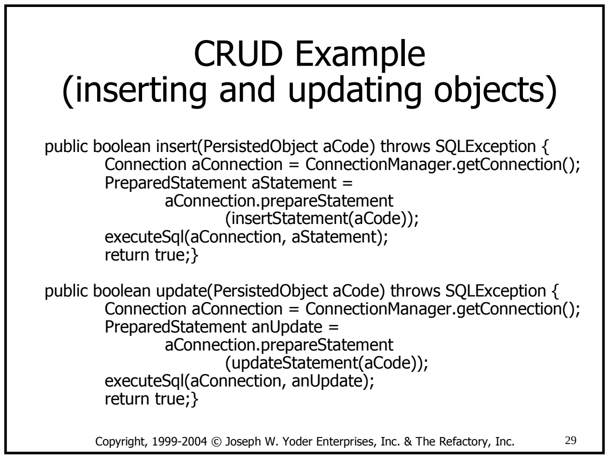
CRUD Example (reading objects)

```
public String selectStatement(PersistedObject anObject) {
    String sql;
    sql = ("SELECT "
        + (TableManager.getQualifiedColumnsFor(brokerFor()))
        + " FROM "
        + (TableManager.getQualifiedTableNameFor(brokerFor()))
        + " A ");

    return(sql + getWhereClause(anObject) + getOrderByClause());
}
```



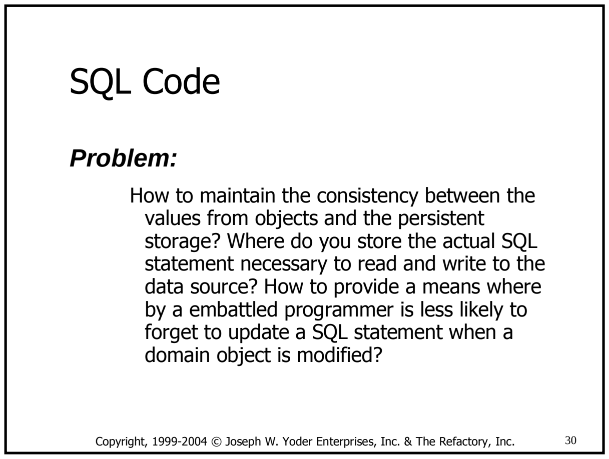
CRUD Example (deleting objects)

```
public boolean delete(String aBroker,PersistedObject aCode) throws
    SQLException {
    Connection aConnection = ConnectionManager.getConnection();
    PreparedStatement aDelete =
                  aConnection.prepareStatement
                                  (deleteStatement(aCode,aBroker));
    executeSql(aConnection, aDelete);
    return true;}

public String deleteStatement(PersistedObject aCode, String aBroker) {
    String sql = ("DELETE FROM "
       + (TableManager.getQualifiedTableNameFor(aBroker))
          + " WHERE OBJECT_ID = "
          + TypeConverter.prepForSql(aCode.getObjectId()));
    return(sql);}
```



CRUD Example (inserting and updating objects)

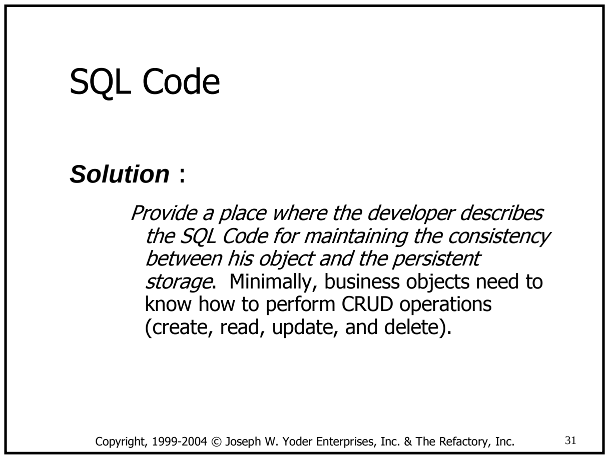
```
public boolean insert(PersistedObject aCode) throws SQLException {
          Connection aConnection = ConnectionManager.getConnection();
          PreparedStatement aStatement =
                    aConnection.prepareStatement
                              (insertStatement(aCode));
          executeSql(aConnection, aStatement);
          return true;}

public boolean update(PersistedObject aCode) throws SQLException {
          Connection aConnection = ConnectionManager.getConnection();
          PreparedStatement anUpdate =
                    aConnection.prepareStatement
                              (updateStatement(aCode));
          executeSql(aConnection, anUpdate);
          return true;}
```



SQL Code

Problem:

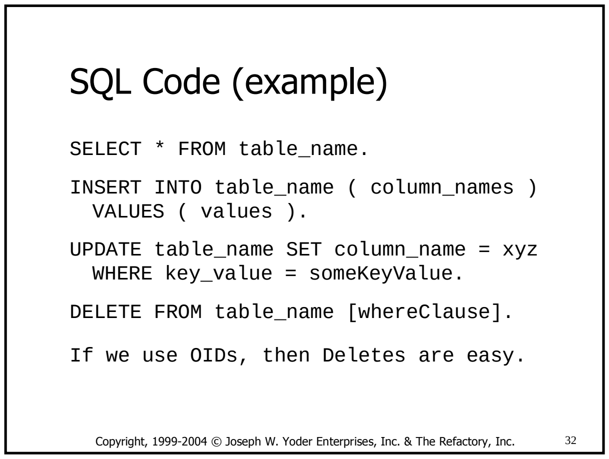
How to maintain the consistency between the values from objects and the persistent storage? Where do you store the actual SQL statement necessary to read and write to the data source? How to provide a means where by a embattled programmer is less likely to forget to update a SQL statement when a domain object is modified?



SQL Code

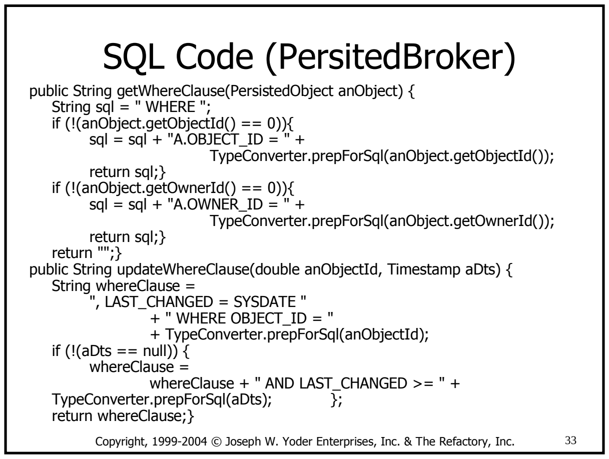
Solution :

Provide a place where the developer describes the SQL Code for maintaining the consistency between his object and the persistent storage. Minimally, business objects need to know how to perform CRUD operations (create, read, update, and delete).



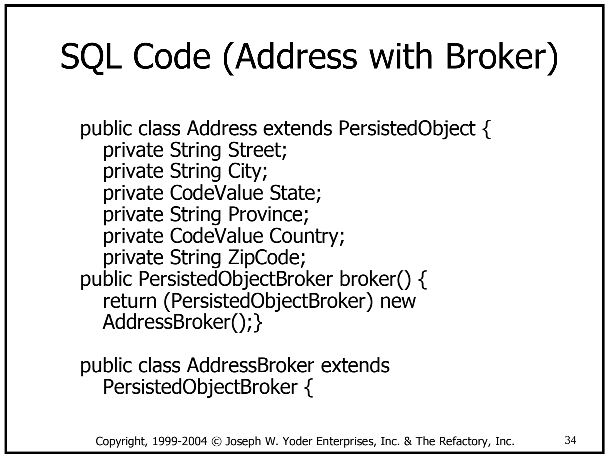
SQL Code (example)

```
SELECT * FROM table_name.

INSERT INTO table_name ( column_names )
  VALUES ( values ).

UPDATE table_name SET column_name = xyz
  WHERE key_value = someKeyValue.

DELETE FROM table_name [whereClause].

If we use OIDs, then Deletes are easy.
```



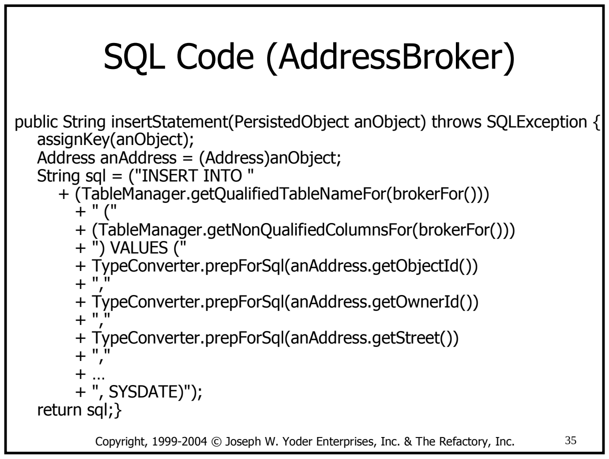
SQL Code (PersitedBroker)

```
public String getWhereClause(PersistedObject anObject) {
    String sql = " WHERE ";
    if (!(anObject.getObjectId() == 0)){
            sql = sql + "A.OBJECT_ID = " +
                        TypeConverter.prepForSql(anObject.getObjectId());
            return sql;}
    if (!(anObject.getOwnerId() == 0)){
            sql = sql + "A.OWNER_ID = " +
                        TypeConverter.prepForSql(anObject.getOwnerId());
            return sql;}
    return "";}
public String updateWhereClause(double anObjectId, Timestamp aDts) {
    String whereClause =
            ", LAST_CHANGED = SYSDATE "
                    + " WHERE OBJECT_ID = "
                    + TypeConverter.prepForSql(anObjectId);
    if (!(aDts == null)) {
            whereClause =
                    whereClause + " AND LAST_CHANGED >= " +
    TypeConverter.prepForSql(aDts);            };
    return whereClause;}
```



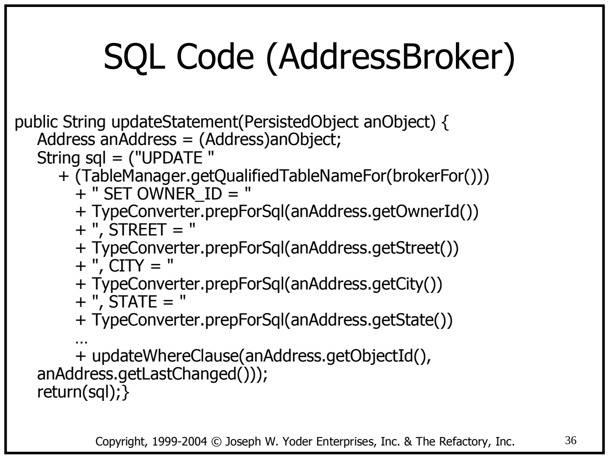
SQL Code (Address with Broker)

```
public class Address extends PersistedObject {
    private String Street;
    private String City;
    private CodeValue State;
    private String Province;
    private CodeValue Country;
    private String ZipCode;
public PersistedObjectBroker broker() {
    return (PersistedObjectBroker) new
    AddressBroker();}

public class AddressBroker extends
    PersistedObjectBroker {
```



SQL Code (AddressBroker)

```
public String insertStatement(PersistedObject anObject) throws SQLException {
    assignKey(anObject);
    Address anAddress = (Address)anObject;
    String sql = ("INSERT INTO "
        + (TableManager.getQualifiedTableNameFor(brokerFor()))
            + " ("
            + (TableManager.getNonQualifiedColumnsFor(brokerFor()))
            + ") VALUES ("
            + TypeConverter.prepForSql(anAddress.getObjectId())
            + ","
            + TypeConverter.prepForSql(anAddress.getOwnerId())
            + ","
            + TypeConverter.prepForSql(anAddress.getStreet())
            + ","
            + ...
            + ", SYSDATE)");
    return sql;}
```



SQL Code (AddressBroker)

```
public String updateStatement(PersistedObject anObject) {
    Address anAddress = (Address)anObject;
    String sql = ("UPDATE "
        + (TableManager.getQualifiedTableNameFor(brokerFor()))
            + " SET OWNER_ID = "
            + TypeConverter.prepForSql(anAddress.getOwnerId())
            + ", STREET = "
            + TypeConverter.prepForSql(anAddress.getStreet())
            + ", CITY = "
            + TypeConverter.prepForSql(anAddress.getCity())
            + ", STATE = "
            + TypeConverter.prepForSql(anAddress.getState())
            ...
            + updateWhereClause(anAddress.getObjectId(),
    anAddress.getLastChanged()));
    return(sql);}
```

Attribute Mapping Methods

Problem:

Where and how does the developer describe the mappings between database values and attributes? When values are brought in from the database, it needs to be defined which attributes the values are mapped to and vice-versa.

37

Attribute Mapping Methods

Solution :

For every domain object that needs to be persistent, create a means to describe the mappings between the database columns and object attributes. In our case, we write a method that describes the mappings from the database values to the object's attributes and write a method which maps the values from the object back to the database.

38

Attribute Mapping Methods

Discussion :

The Persistence Layer will use this method to take the returned values from the database and stored them in the appropriate object attribute. Similarly, when the persistent object is being saved, the Persistent Layer will use a similar method for taking values from the object and putting them out to the database.

39

Attribute Mapping (Address)

```
public PersistedObject initializeFrom(java.sql.ResultSet aRow) throws
    java.sql.SQLException {
    Address anAddress = new Address();
    anAddress.setObjectId(TypeConverter.convert
                                (aRow.getDouble("OBJECT_ID")));
    anAddress.setOwnerId(TypeConverter.convert
                                (aRow.getDouble("OWNER_ID")));
    anAddress.setStreet(TypeConverter.convert
                                (aRow.getString("STREET")));
    anAddress.setCity(TypeConverter.convert(aRow.getString("CITY")));
    …
    anAddress.setZipCode(TypeConverter.convert
                                (aRow.getString("ZIPCODE")));
    anAddress.setLastChanged(TypeConverter.convert
                                (aRow.getTimestamp("LAST_CHANGED")));
    return (PersistedObject)anAddress;}
```

40

Attribute Mapping (example)

```
public String insertStatement(PersistedObject anObject)
    throws SQLException {
    assignKey(anObject);
    Address anAddress = (Address)anObject;
    String sql = ("INSERT INTO "
        + (TableManager.getQualifiedTableNameFor(brokerFor()))
           + " ("
           + (TableManager.getNonQualifiedColumnsFor(brokerFor()))
           + ") VALUES ("
           + TypeConverter.prepForSql(anAddress.getObjectId())
           + ","
           + TypeConverter.prepForSql(anAddress.getOwnerId())
           + ","
           + TypeConverter.prepForSql(anAddress.getStreet())
           + "," …
```

41

Type Conversion

Problem:

There is an impedance mismatch between RDB-types & object-types. How do we take objects that may not have a type in a database and allow for them to map to a database type? How do we ensure the data read from the data source will work with our object? How do we ensure the data written to the data source will comply with the data source's rules and maintain data integrity?

42

Type Conversion

Solution :

Have all values convert their respective types through a Type Conversion object. This object knows how to handle nils and other mappings of objects to and from database values. When objects are persisted from large multi-application data source the data formats can vary. This pattern ensures the data retrieved from the data source is appropriate for the object. This can also be a *Strategy* for pluggable type converters

43

Type Conversion (Java example)

```
public class TypeConverter extends AbstractManager{
public static boolean convert(boolean aBoolean){
    return aBoolean;}
public static Date convert(java.sql.Date aDate){
    return aDate;}
public static String convert(String aString){
    if (aString == null ) { return null;}
    return aString.trim();}
public static Timestamp convert(Timestamp aDts){
    return aDts;}
```

44

Type Conversion (example)

```
public static String prepForSql(double aNumber) {
    if (aNumber == 0){
        return "NULL" ;}
    Double adouble = new Double(aNumber);
    String sqlValue = adouble.toString();
    return sqlValue;
}

public static String prepForSql(CodeValue aCode) {
    if (aCode == null){
        return "NULL";}
        String aValue =
    Integer.toString(prepForSql(aCode.getCodeValue()));
    return aValue;
}
```

45

Type Conversion (example)

```
public static String prepForSql(java.sql.Date aDate) {
    if (aDate == null) {return null;}
    String aString =
    java.text.DateFormat.getDateInstance(1).format(aDate);
    String sqlDate = "TO_DATE('"
                                    + aString
                                    + "','MONTH DD YY')";
    return (sqlDate);}

public static String prepForSql(java.util.Date aDate) {
    if (aDate == null) {return null;}
    String aString =
    java.text.DateFormat.getDateInstance(1).format(aDate);
    String sqlDate = "TO_DATE('"
                                    + aString
                                    + "','MM DD YYYY')";
    return (sqlDate);}
```

46

Changed Manager

Problem:

Many objects need access to shared values, but the values are not unique throughout the system. How to tell that an object has changed and needs to be saved to the data source? How to prevent unnecessary access to the data source?

47

Changed Manager

Solution :

Create a means to keep track of what objects are dirty so you can insure to save these changes when desired. You can inherit from a Persistent *object, which has a dirty bit that gets set whenever one of its attributes that maps to the database is changed.* This dirty bit is usually an instance variable with a boolean value which indicates when an objects values have changed.

48

Changed Manager

Discussion :

When the boolean value is set the Persistent object will save the new values to the data source. If the boolean value is not set the Persistent object will bypass the write to the data source. Dependent upon the class hierarchy the implementation can vary. One solution is to modify the setter methods to set the flag whenever an object's values are changed.

Changed Manager

Other ways are to either keep the original values and compare before the save,

Or, put the dirty objects into a bag and use this to know which objects are dirty when saving is needed.

OID Manager (Key Manager)

Problem:

How do we insure that each object gets stored uniquely in a database regardless if it shares similar state with another object or not?

OID Manager

Solution :

Provide a Object Identity Manager that creates unique keys for all objects that need to be stored in the database. Insure that all newly created objects that need to be persisted get a unique key. When a new object that needs to be persisted is to be written to the data source a unique identifier is generated. The generation process needs to be quick and ensure uniqueness.

OID Manager

Discussion :

The OID Manager is usually an object that just encapsulates the key generation algorithm. The *OID Manager* can use a *Strategy* to generate its unique key.

Transaction Manager

Problem:

How do we group together the saving of multiple objects in such a way that if the saving of one object fails, then we want the other objects to not be saved?

Transaction Manager

Solution :

Build a Transaction Manager that works similar to other transactions managers. This manager allows for the beginning of transactions, the ending of transactions, the committing of transactions, and the rollback of transactions. The transaction manager usually maps to the RDBMS's transaction manager.

Connection Manager

Problem:

How does the persistent manager keep track of the database to connect to and what user is currently connected?

Connection Manager

Solution :

Create a Connection Manager object, which holds all of the values that need to be used for the database connection. The common values are usually the database session, the current user logged into the system, and any other global information used for auditing, transactions, and the like.

Connection Manager

Discussion :

The Connection Manager establishes the connections to the databases. A *Strategy* can be used for deciding which connection is needed if multiple database servers are being used. The Connection Manager can use a *Session*.

Connection Manager (example)

```
public class ConnectionManager
   extends AbstractManager{
   private static java.sql.Connection activeConnection;
```

Connection Manager (example)

```
private static java.sql.Connection
   getActiveConnection() {
   return activeConnection;}

public static void closeConnection(Connection
   aConnection) throws java.sql.SQLException {
   aConnection.commit();
   aConnection.close();
   setActiveConnection(null);}
```

Connection Manager (example)

```
public synchronized static Connection getConnection()
    throws SQLException {
    if (getActiveConnection() != null){
          return getActiveConnection();}
    DriverManager.registerDriver(new
                          oracle.jdbc.driver.OracleDriver());
//  DriverManager.registerDriver (new
                          sun.jdbc.odbc.JdbcOdbcDriver());
    Connection aConnection =
                   DriverManager.getConnection(
//                        "jdbc:oracle:thin:" + getServerSpecs() +
    …
    aConnection.setAutoCommit(false);
    setActiveConnection(aConnection);
    return aConnection;}
```

61

Connection Manager (example)

```
private static String getServerSpecs() {
   String serverSpecs = new String();
   try { FileReader fileInStream;
        BufferedReader dataInStream;
        String configFile = System.getProperty("SFPS_DB_CONFIG_FILE");
        System.out.println("configfile: " +configFile);
        if(configFile == null)
        {configFile = "/export/home/sfps.init";}
        fileInStream = new FileReader(configFile);
        dataInStream = new BufferedReader(fileInStream);
        serverSpecs = dataInStream.readLine();
        fileInStream.close();
        dataInStream.close();}…
```

62

Connection Manager (example)

```
   …catch (FileNotFoundException noFile)
   {System.out.println("Database server spec. file ("
         + noFile.getMessage() + ") not found, using defaults.");
//Defaults
//        serverSpecs ="@192.168.100.200:1521";
          serverSpecs = "@192.168.100.200:1521" + ":sfps1";
//        serverSpecs = "@ISS7M5Z501:1521";
   } catch (IOException notReadable)
   {System.out.println("Database server spec. file ("
   + notReadable.getMessage()  + ") could not be read, using defaults.");
//Defaults
   serverSpecs ="@192.168.100.200:1521" + ":sfps1";
//  serverSpecs = "@ISS7M5Z501:1521";}
   return serverSpecs;}
```

63

Mapping Objects To RDBMS Persistence Pattern Language

- Persistence Layer
- CRUD
- SQL Code
- Attribute Mapping Methods
- Type Conversion

64

Mapping Objects To RDBMS Persistence Pattern Language

- Changed Manager
- OID Manager
- Transaction Manager
- Connection Manager

65

Summary of Architecture

- Subclass from PersistedObject and Subclass from PersistedBroker
- Overwrite:
  - TableName
  - Initialize Method
  - Create, Update, Read
- Much simpler and easier to understand and maintain

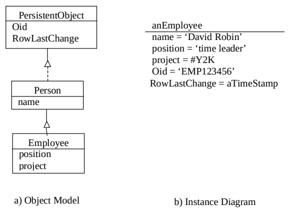
66

Mapping Objects (the relational side)

- Object Identification
  - uniquely generated
  - "HIGH/LOW OID" presented in [Ambler98].
- Classes to Tables [Brown Whitenack 96].
- Handling Related Objects
  - owned objects
  - knowledge relationships
  - many-to-many
- Optimizations (denormalization)

67

One Table for each Concrete Class

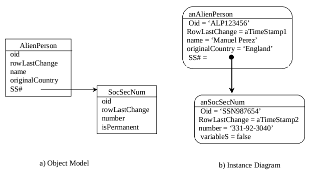


a) Object Model b) Instance Diagram

oid	ownerOid	rowLastChange	name	position	project
EMP123456	null	aTimeStamp	'David Robin'	'time leader'	#Y2K

68

Owned Objects



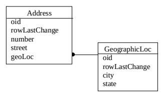
a) Object Model b) Instance Diagram

oid	ownerOid	rowLastChange	name	originalCountry
ALP123456	null	aTimeStamp1	'Manuel Perez'	'England'

oid	ownerOid	rowLastChange	variableT	variableS
SSN987654	ALP123456	aTimeStamp2	'331-92-3040'	false

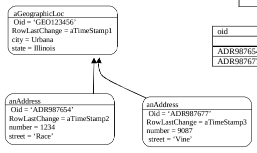
69

Knowledge Relationships



a) Object Model

oid	ownerOid	rowLastChange	city	state
GEO123456	null	aTimeStamp1	Urbana	Illinois

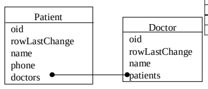


oid	ownerOid	geoLocOid	rowLastChange	number	street
ADR987654	null	GEO123456	aTimeStamp2	1234	Race
ADR987677	null	GEO123456	aTimeStamp3	9087	Vine

b) Instance Diagram

70

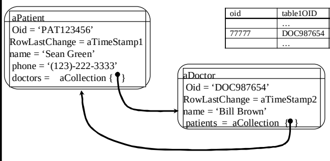
Many to Many Relationships



a) Object Model

oid	ownerOid	rowLastChange	name	phone
PAT123456	null	aTimeStamp1	Sean Green	(123)-222-3333

oid	ownerOid	rowLastChange	name
DOC987654	null	aTimeStamp2	Bill Brown



oid	table1OID	table2OID	table1ClassType	table2ClassType	rowLastChanged
	…	…			…
77777	DOC987654	PAT123456	Doctor	Patient	TimeStamp*N*
	…	…			…

b) Instance Diagram

71

Mapping Objects (meta-architecture)

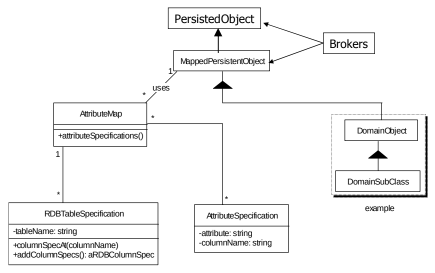
- Most of the time the only difference between the SQL generated is the table names, column names, and the rdb-types, attribute names, object-types
- When creating CRUD there are many places where the code looks very similar
- Can use a specification to parameterize the differences between objects

72

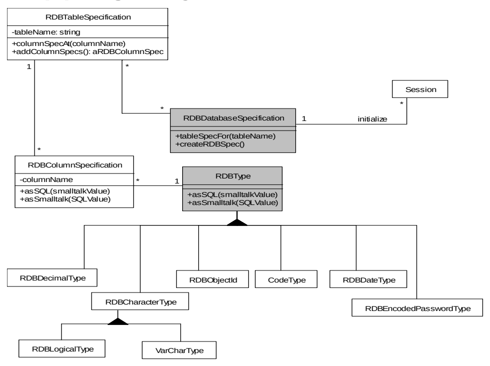
SQL Code (example)

```
SELECT * FROM table_name.

INSERT INTO table_name ( column_names )
  VALUES ( values ).

UPDATE table_name SET column_name = xyz
  WHERE key_value = someKeyValue.

DELETE FROM table_name [whereClause].
```

Mapping Objects (meta-architecture)



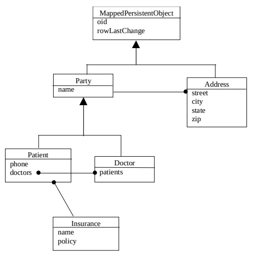
Mapping Objects (meta-architecture)



Attribute Map Specification

- Specification for the attribute map needs
  - Table Name(s)
  - Attribute Names
  - Column Names
  - RDB Types with Parameters
  - Type of Relationship

Attribute Mapping Example



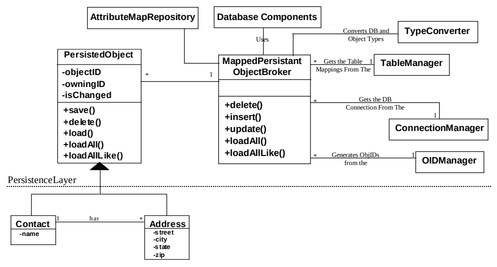
Class Diagram

Attribute Map Example

```
(PATIENT_T
   (name NAM_STR (RDBVarChar 20) (simple string))
   (address Address (RDBObjectId) owned)
   (phone PHONE_STR (RDBVarChar 10) (simple string))
   (doctors Doctor (RDBManyType) many)
   (insurance Insurance (RDBKnowledgeType INS_OID)
                                          knowledge))
```

This spec could be XML or whatever…it is used to generate the SQL Code anytime a CRUD operation is executed

Meta-Architecture



79

Query Objects

Making Objects to deal with Queries

By: John Brant and Joseph Yoder

Published in PLoPD4 Book

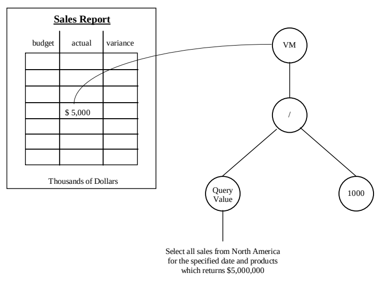
80

Query Objects (Hot Spots)

- Find aspects that change, and make them objects
- Often are patterns from *Design Patterns: Elements of Reusable Object-Oriented Software*
- QueryObjects: Interpreter pattern

81

Typical Values in a Report


Sales Report
budget
actual
variance
$ 5,000
Thousands of Dollars
VM
/
Query Value
1000
Select all sales from North America for the specified date and products which returns $5,000,000

82

Query Objects

Business logic is equations expressed with Values as Mathematical functions and Queries

- Values = functions of other values
- Values = queries from the database

variable margin = net sales - variable cost

net sales = gross sales - warrantee

gross sales = sum *sales* column from *sales_and_transfer* table

83

Interpreter Pattern

- Need to represent SQL to manipulate query:

  SELECT SUM(sales) FROM sales_and_transfer

  WHERE family='MWL' AND date > '1/1/96' AND date < '1/1/97'

- Problem: how do you represent a simple language?

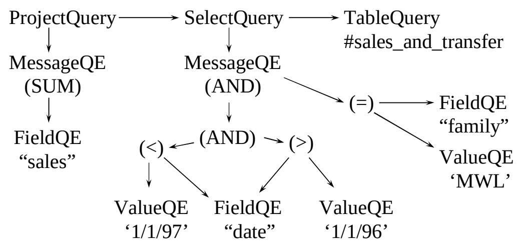
84

Interpreter Pattern

1) make a class hierarchy that represents nodes in abstract syntax tree (SELECT, AND, <, tables, field names)
2) define methods to construct and manipulate tree
3) define method to compute value of query (this is the "interpreter")

Instance Hierarchy

SELECT SUM(sales) FROM sales_and_transfer
WHERE family='MWL' AND date > '1/1/96' AND date< '1/1/97'



QueryObjects

- QueryObject
  - TableQuery
  - JoinQuery
  - WrapperQuery
    - RenamingQuery
    - ExpressionQuery
      - SelectQuery
      - ProjectQuery
      - OrderQuery

- QueryExpression
  - ValueQE
  - MessageQE
  - FieldQE
    - RenamedFieldQE

QueryObject Protocol

- values - answer collection of tuples
- fieldNames
- join: aQueryObject
- select: aQueryExpression
- project:, renameColumnsTo:, outerJoin:, groupBy:, orderBy:, asDistinct

Creating a QueryObject

```
salesQ := #sales_and_transfer asQuery.
dateQ := salesQ select:
        ((salesQ @@ 'family') = 'MWL') &
        ((salesQ @@ 'date') > '1/1/96') &
        ((salesQ @@ 'date') < '1/1/97').
dateQ project: (dateQ @@ 'sales') Sum
```

QueryExpression Protocol

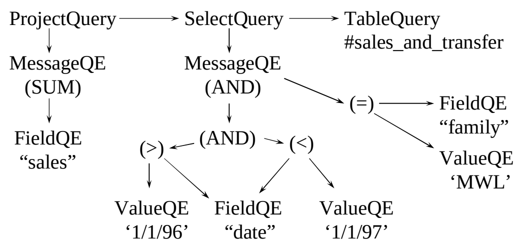
+, -, <, =, &, |, Sum, Average, Count, …

Sending one of these messages to a QueryExpression builds a MessageQE with the appropriate operands, and with the message as the operator.

Observer and QueryObjects

Let ValueQE refer to a ValueModel.

Let each QueryObject observe its components.



91

Summary

- A Persistent Layer helps hide database technology details from the application developer and makes it easier to change persistent storage technologies without affecting the application code.
- Brokers are useful for separating SQL details from domain objects.

92

Summary

- Patterns are good for documenting a framework and for describing how to build a similar framework.
- This pattern language follows what one needs to do when dealing with persisting objects in a non-object world.

93

Related Links

- The following link discusses the details of the framework
  http://www.joeyoder.com/Research/objectmappings
- Generic Lightweight Object-Relational Persistence (GLORP)
  http://www.glorp.org/
- Joe's Patterns Paper
  http://www.joeyoder.com/papers/patterns
- The reporting patterns describing query-objects - PLoP '96
  http://www.joeyoder.com/papers/patterns/Reports/
- Evolving Frameworks - PLoP '97
  http://st-www.cs.uiuc.edu/users/droberts/evolve.html

94

Related Links

- Security Patterns describing Sessions - PLoP '97
  http://www.joeyoder.com/papers/patterns/
- Crossing Chasms
  http://www.ksccary.com/ordbjrnl.htm
- Mapping Object to Relational Databases
  http:// www.ambysoft.com/mappingObjects.pdf
- Relational Database Access Layers
  http://www.sdm.de/g/arcus/cookbook/relzs/
- Metadata and Adaptive Object-Models
  http://www.adaptiveobjectmodel.com
- HOP: Persistency Framework
  http://www.elevensoft.it/hoop

95

That's All

96