Object-Oriented Databases
Commercial OODBMS: Part 1

- Objectivity/DB for C++
- ObjectStore PSE Pro for Java
Persistence Strategies

- Persistence by inheritance
  - persistence capabilities inherited from pre-defined persistent class
  - Versant (C++), Objectivity/DB (C++)

- Persistence by instantiation
  - objects made persistent and get persistence capabilities upon instantiation
  - ObjectStore (C++)

- Persistence by reachability
  - objects made persistent if reachable from other persistent object
  - O₂ (C++/Java), ObjectStore (Java), Versant (Java/Smalltalk), Objectivity/DB (Java/Smalltalk), db4o (Java/.NET), ODMG
Objectivity/DB

- Object-oriented database management system
  - developed since 1993 by Objectivity, Inc.
  - version 9.3 released in October 2006
- Contributions in associations and version management
- Originally based on C++
  - extended C++ with proprietary concepts
  - implemented using a special pre-processor
- Now has
  - Smalltalk API
  - Java API
  - Python API
  - SQL++ interface
Objectivity Product Family

<table>
<thead>
<tr>
<th>Administration Tools</th>
<th>C++</th>
<th>Java</th>
<th>Smalltalk</th>
<th>SQL++</th>
<th>ODBC Driver</th>
<th>Fault Tolerant Option</th>
<th>Data Replication Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objectivity/DB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Database Servers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Extensions to C++

- Create and delete objects
- Maintain and navigate associations between objects
- Access functions
  - name and lookup objects, map dictionaries for indexing objects
  - iterators over object collections
  - query and access objects using the SQL++ interface
- Version functions
  - create and locate versions of objects
  - track version genealogy
- Copy and move functions
  - copy and move objects between containers
C++ Application Development Process

- Design schema and create data model files in DDL
- Process data model files using the DDL processor
- Complete application source code
- Compile C++ application source files and data model source files generated by DDL processor
- Link compiled code with Objectivity/C++ runtime libraries
- Supported by provision of a Makefile
C++ Application Development Process

DDL Schema Code

DDL Processor

Schema Source Code

Schema Header

Objectivity Runtime Library

Objectivity Federated Database

Application Code

C++ Compiler

Linker

Objectivity Application
Storage Classes

- **Basic object**
  - basic objects implement `com.objy.db.iapp.Persistent`
  - persistence by inheriting from `com.objy.db.app.ooObj`

- **Container**
  - collection of basic physically clustered objects

- **Database**
  - collection of containers
  - comprises default container plus user-defined containers

- **Federated database**
  - contains user-defined databases plus schema
Logical Storage Model

Federated Database

User-Defined Database

Basic Objects

User-Defined Containers

Default Containers
Persistence-Capable Classes

- Every persistent instance of a persistence-capable class has an associated internal object called a persistor
  - contains internal state and implements persistence behavior
  - created when transient object made persistent and when existing persistent object is retrieved
  - any operation on a persistent object is performed by calling the appropriate method of the object’s persistor

- Can support four general kinds of persistence behavior
  - implicit persistence behavior
  - explicit persistence
  - handle persistent events
  - automatic persistence with relationships
Inheriting Persistence Behaviour

- Implicit persistence behaviour
  - get and set methods required
  - default handling for persistent events (activate, deactivate, pre-write...)

- Non-static, non-transient fields
  - numeric attributes
  - string attributes
  - date or time attributes
  - references to other instances of persistent-capable classes
  - arrays of the above types

```java
import com.objy.db.app.ooObj;
public class Author extends ooObj {
    private String name;
    private Date birthdate;
    public Author(String name) {
        this.name = name;
        this.birthdate = null;
    }
    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }
    ...
}
```
Containers

- Physical grouping of basic objects in database
- Third highest level in logical storage hierarchy
- Two different kinds of containers
  - garbage-collectible: deleted objects are removed automatically
  - non-garbage-collectible: deleted objects are not removed
- Functionality
  - locking of objects
  - indexes
  - object caching
  - retrieving objects
  - clustering objects
Containers

```java
Connection connection = Connection.open("fdb", oo.openReadWrite);
Session session = new Session();
session.begin();
ooFDObj fedDb = session.getFD();
if (fedDb.hasDB("PublicationsDB")) {
    ooDBObj pubDb = fedDb.lookupDB("PublicationsDB");
    moiraPubs = new com.objy.db.app.ooContObj();
    // Make the container persistent by adding it to a database
    pubDB.addContainer(moiraPubs, "MoirasPublications", 0, 5, 10);
    // Make article persistent by clustering it in a container
    Article a = new Article("Document Profiling to Enhance Collaboration");
    moiraPubs.cluster(a);
} else {
    session.abort();
    return;
}
session.commit();
```

*Note: exception handling has been omitted!*
Objectivity/DB Associations

- Supports associations between objects
  - declared within classes
  - unary and binary associations
  - binary associations represented internally as separate construct
- Mutual consistency of relationships is maintained
- Can attach semantics with associations
  - delete and lock propagations
  - copying behaviour
  - versioning behaviour
Operation Propagation along Associations

- Delete and lock operations can be propagated along associations.
**Defining Associations**

```java
public class Author extends ooObj {
    
    private ToManyRelationship publications;
    
    public static ManyToMany publications_Relationship() {
        return new ManyToMany(
            "publications", // relationship field
            "ch.ethz.globis.demo.Publication", // destination class
            "authors", // inverse relationship
            Relationship.COPY_MOVE, // copying behavior
            Relationship.VERSION_MOVE, // versioning behavior
            false, // delete propagation
            false, // lock propagation
            Relationship.INLINE_NONE // storage method
        );
    }
}
```
Persistent Collections and Iterators

- Built-in persistent collections provide sets, lists and maps
  - ordered vs. unordered and scalable vs. non-scalable
  - implement interface `com.objy.db.util.ooCollection`
  - `ooTreeListX`, `ooHashSetX`, `ooTreeSetX`, `ooMap`, `ooHashMapX` and `ooTreeMapX`

- Object iterators
  - step through a group of objects found in the federated database
  - containers

- Scalable-collection iterators
  - step through the objects in a scalable persistent collection
Retrieving Objects

- Creating and following links
- Individual and group lookup of persistent objects
  - through keys and iterators
- Parallel query
  - Parallel Query Engine (Objectivity/PQE)
  - divides the query scope among a number of query servers
- Content-based filtering
  - predicate-query language supporting primitive types and strings
  - used in predicate scans in group lookups
  - used when following a to-many relationship
  - used to find destination objects in parallel queries
ObjectStore Product Family

- Both Java and C++ environments supported
- ObjectStore Personal Storage Edition (PSE) Pro
  - pure Java-based lightweight object database
  - large, single-user databases
  - small memory footprint (~500kB)
  - multithreaded
  - embedded systems, mobile computing and desktop applications
- ObjectStore Enterprise
  - high-performance, distributed, multi-user database
  - distributed, persistent, transactional object caching
  - clustering, online backup, replication, high availability
- Migration of applications to from PSE to Enterprise easy
Model of Persistence

- Persistence by reachability
  - database roots

```java
Author moira = new Author("Moira C. Norrie");
db.createRoot("Authors", moira);
```

- Persistence capable classes
  - post-processor makes classes persistent capable

- Persistent aware classes
  - can access and manipulate persistent objects but are not themselves persistent
Persistent Objects

- After being loaded into memory, persistent objects can be in one of the following three states:
  - **hollow**: used as a proxy and loaded on demand (lazy loading)
  - **active**: when the object is loaded into memory its flag is set to clean, after an update the flag becomes changed to dirty
  - **stale**: no longer valid (e.g. after a commit)
Persistent Capable Classes

- Class `java.lang.String`
- Wrapper classes from the `java.lang` package
  - `Boolean`, `Byte`, `Character`, `Double`, `Float`, `Integer`, `Long` and `Short`
- Arrays of primitive and of any persistence-capable types
Collections

Map

OSTreeMap

OSTreeMapByteArray

OSTreeMapDouble

OSTreeMapFloat

OSTreeMapInteger

OSTreeMapLong

OSTreeMapString

Set

OSAbstractSet

OSHashSet

OSTreeSet

Oktober 24, 2008 Michael Grossniklaus – Department of Computer Science – grossniklaus@inf.ethz.ch 23
Collections

Collection

List

OSDictionary

OSHashtable

Map

OSHashMap

OSHashBag

OSVector

OSVectorList
Post-Processing

1. Compile all source files

   javac *.java

2. Post-process the class files → annotated versions of the class files

   osjcfp -dest . -inplace *.class

3. Run the post-processed main class

   java mainClass
IPersistent

- Post-processing automatically annotates the class to implement the interface `com.odi.IPersistent`

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>clearContents</strong>()</td>
</tr>
<tr>
<td>Resets the values of the fields of an active persistent object to the default initial values of that class of object.</td>
</tr>
<tr>
<td><strong>flushContents</strong>(GenericObject genObj)</td>
</tr>
<tr>
<td>Stores the values of the fields of an active persistent object in an instance of the GenericObject.</td>
</tr>
<tr>
<td><strong>initializeContents</strong>(GenericObject genObj)</td>
</tr>
<tr>
<td>Initializes the values of the fields of a hollow persistent object from data contained in an instance of GenericObject.</td>
</tr>
<tr>
<td><strong>ODIgetRef</strong>()</td>
</tr>
<tr>
<td>Returns the value of the ODIRef field.</td>
</tr>
<tr>
<td><strong>ODIsetState</strong>()</td>
</tr>
<tr>
<td>Returns the value of the ODIOBJECTSTATE field.</td>
</tr>
<tr>
<td><strong>ODIsetRef</strong>(com.odi.imp.ObjectReference objRef)</td>
</tr>
<tr>
<td>Updates the value of the ODIRef field.</td>
</tr>
<tr>
<td><strong>ODIsetState</strong>(byte state)</td>
</tr>
<tr>
<td>Updates the value of the ODIOBJECTSTATE field.</td>
</tr>
</tbody>
</table>
Required Steps

- Create a session
  
  \[\text{Session}#\text{create}(\text{String host, java.util.Properties properties})\]

- Join a thread to a session
  
  \[\text{Session}#\text{join}()\]

- Create or open a database
  
  \[\text{Database}#\text{create}(\text{String name, int fileMode})\]
Required Steps

- **Start and commit transactions**
  
  ```
  Transaction#begin(int type)
  Transaction#commit(int retain)
  ```

- **Create database roots**
  
  ```
  Database#createRoot(String name, Object object)
  ```

- **Ending the session and closing the database**
  
  ```
  Session#terminate()
  ```
Object Deletion

- In ObjectStore objects are deleted by
  - remove object from all structures accessible from database roots
  - persistent garbage collector automatically removes non-reachable objects
- No defragmentation results
Queries

- Queries can only be defined over collections
- Predicate expressions are used

```java
Query query =
    new Query(Author.class, "getName() == "Moira C. Norrie\"");
```

- Query evaluation

```java
Collection result = query.select(authors);
```
Relationships

- Java Dynamic Data classes (JDD)
  - create, store and access persistent data, based on schema
  - defining types and their attributes
  - creating entities of a type
  - querying types
  - creating indexes

- JDD introduces bi-directional relationships
  - one-to-one, one-to-many, many-to-many
  - referential integrity is maintained
  - relationships need to be defined at runtime
Literature

- Objectivity/DB
  - http://www.objectivity.com/
- ObjectStore
  - http://www.progress.com/objectstore/
Next Week
Commercial OODBMS: Part 2

- Versant Object Database for Java
- OODBMS Architectures, Revisited and Defended