Object-Oriented Analysis and Design
Appendix: ObjectStore Mechanism
The following slides can be inserted during the Identify Design Mechanisms module.
Example: Persistency: OODBMS: ObjectStore

Example of a persistent class client

Create to support persistence

Example of a persistent “root” class

All are roles to be filled by the designer applying the mechanism

<<role>>
PersistencyClient

0..* 1

<<role>>
SampleDBManager

initialize()
command()
shutdown()
newPersistentClass() : PersistentClass
removePersistentClass(theClass : PersistentClass)
getPersistentClassData() : PersistentClass

<<role>>
PersistentClass

getData()
setData()
command()
new()
Example: Persistency: OODBMS: ObjectStore: SampleDBManager

SampleDBManager is the application’s interface with the ObjectStore database

ObjectStore
  destroy()

Session
  create()
  join()
  terminate()

Map
  put(name : string, object : Object)
  get(name : string) : Object
  remove(name : string)
  new()

Database
  open()
  create()
  createRoot(name : string, object : Object)
  close()

Transaction
  begin()
  commit(retain : int)

<<role>>
SampleDBManager
  initialize()
  command()
  shutdown()
  newPersistentClass() : PersistentClass
  removePersistentClass(theClass : PersistentClass)
  getPersistentClassData() : PersistentClass

1
1
1
1
1
1
1
1
0..*
1

1
1
1
1
1
1
1
Example: Persistency: OODBMS: ObjectStore: Initialize

1. initialize()
   1.1. create()
   1.2. join()
   1.3. create()
   1.4. begin()
   1.6. createRoot(string, Object)
   1.7. commit()

Pass in the name of the Map, as well as the Map Itself (the Map is an Object).

The root is the entry point into the Database. It is a "special" data structure. Any changes to this data structure that occur within the context of a transaction will be applied to the associated ObjectStore Database.

Create the Map that will serve as the database root.
Example: Persistency: OODBMS: ObjectStore: Create

1: newClass()
2: begin()
3: new()
4: put(string, Object)
5: commit()

The root is the entry point into the database.

Pass in the unique key for the PersistentClass and the PersistentClass. The PersistentClass will be stored in the Map, and subsequently, in the ObjectStore Database.
Example: Persistency: OODBMS: ObjectStore: Read

1. getPersistentClassData()
   1.1. begin()
   1.2. get(string)
   1.3. getData(String)
   1.4. commit(RETAIN_HOLLOW)

Start a read-only transaction to ensure that the object isn't changed while we're reading it.

Find the object in the database; pass in the unique key.

Specify the RETAIN_HOLLOW option on the commit(), so that the references to the retrieved data can be used outside of the transaction.

Read the data from returned object.

The root is the entry point into the Database.

PersistencyClient
SampleDBManager
Transaction
PersistentClass
DatabaseRoot: Map

Object Oriented Analysis and Design
Example: Persistency: OODBMS: ObjectStore: Update

1: command()
2: begin()
3: get(string)
4: command()
5: commit

The root is the entry point into the Database
Find the object in the database; pass in the unique key
invoke the object command
Example: Persistency: OODBMS: ObjectStore: Delete

PersistencyClient

: Transaction

DatabaseRoot

: Map

: PersistentClass

: ObjectStore

1. removePersistentClass(PersistentClass)

1.1. begin()

1.2. get(string)

1.3. getData()

1.4. remove(string)

1.5. destroy(Object)

1.6. commit

If the instance to be deleted has constituent parts (i.e., there's a composition between their associated classes), those parts must be retrieved and manually deleted. This is necessary as it is not always the case that the parts are removed with the whole (only with composition are the lifetimes coincident).

Retrieve the instance to be removed

Once the "parts" have been removed, remove the original class

Completely remove the instance from the ObjectStore database. This syncs up the cache and the physical database. This forces the delete to happen right away, as opposed to being cached (you don't want something else to link to an object that is to be deleted).

The root is the entry point into the Database.
Example: Persistency: OODBMS: ObjectStore: Shutdown

1: shutdown( )
2: close( )
3: terminate( )
Incorporating ObjectStore: Steps

- Provide access to the class libraries needed to implement ObjectStore access
  - *Dependency on com.odi*

- Select the database root class(es)
  - *Student class*

- Select the container class(es) that will serve as the database root(s) (contains the selected root class(es))
  - *Map (from com.odi)*
  - *Key will be Student ID*
Create a DBManager (one per ObjectStore database instance)

- **Single Course Registration Database** => **CourseRegDBManager**
- **Will “live in” ObjectStore Support package**

Add operations to DBManager to access entities in the OODBMS

- **Create operations for Student and Schedule**

Create/Update interaction diagrams that describe:

- Database initialization and shutdown
- Persistent class access: Create, Read, Update, Delete

Implement persistent classes

- Add “import com.odi.*” statement
- **Implementer to include this statement**
Example: Incorporating ObjectStore

CourseRegDBManager

- initialize()
- shutdown()
- newSchedule(forStudent : Student) : Schedule
- delete(theSchedule : Schedule, forStudent : Student)
- update(theSchedule : Schedule, forStudent : Student)
- save(theSchedule : Schedule, forStudent : Student)
- getSchedule(forStudent : Student, forSemester : Semester) : Schedule
- newStudent() : Student
- delete(theStudent : Student)
- update(theStudent : Student)
- save(theStudent : Student)
- getStudent(withID : string) : Student

Session

- create()
- join()
- terminate()

Map

- put()
- get()
- remove()
- new()

Database

- open()
- create()
- createRoot(name : string, object : Object)
- close()

ObjectStore

- destroy()

Transaction

- begin()
- commit(retain : int)

Student

- studentID : int

Schedule

- addSchedule()
- getSchedule()
- deleteSchedule()
Example: Incorporating ObjectStore (cont.)

ObjectStore Support

- CourseReg
- DBManager
  (from ObjectStore Support)

University Artifacts

com.odi

- Map
  (from com.odi)
- Session
  (from com.odi)
- Transaction
  (from com.odi)
- Database
  (from com.odi)
Use-Case Design Slides

The following slides can be inserted during the Use-Case Design module
Incorporating ObjectStore: Steps

- Provide access to the class libraries needed to implement ObjectStore access
  - Dependency on com.odi

- Select the database root class(es)
  - Student class

- Select the container class(es) that will serve as the database root(s) (contains the selected root class(es))
  - Map (from com.odi)
  - Key will be Student ID

- Done (continued)
Create a DBManager (one per ObjectStore database instance)

- Single Course Registration Database => CourseRegDBManager
- Will “live in” ObjectStore Support package

Add operations to DBManager to access entities in the OODBMS

- Create operations for Student and Schedule

Create/Update interaction diagrams that describe:
- Database initialization and shutdown
- Persistent class access: Create, Read, Update, Delete

Implement persistent classes
- Add “import com.odi.*” statement
- Student to include this statement

Out of scope

√ - Done
Example: Incorporating OODBMS Persistency: Initialize

1. initialize()
   1.1. create()
   1.2. join()
   1.3. create()
   1.4. begin()
   1.5. new()
   1.6. createRoot("StudentMap", StudentMap)
   1.7. commit()

Pass in the name of the Map, as well as the Map itself (the Map is an Object).

Create the StudentMap that will serve as the database root.

The StudentMap is the entry point into the Database. It is a "special" data structure. Any changes to this data structure that occur within the context of a transaction will be applied to the associated ObjectStore Course Registration Database.
Example: Incorporating OODBMS Persistency: Shutdown

RegistrationController

CourseRegDBManager

CourseRegDB

Database

Session

1. shutdown( )

1.1. close( )

1.2. terminate( )
Example: Incorporating OODBMS Persistency: Create/Update

RegistrationController : CourseRegDBManager : Transaction

Passed In : Student.
: Map
Retrieved from Database : Student.

1. save(Schedule, Student)
   1.1. begin()
   1.2. // getStudentID()
   1.3. get (StudentID)
   [ Student does not exist]
   1.4. put(StudentID, Student)
   [ Student does exist]
   1.5. // addSchedule(Schedule)
   1.6. commit()

- A read-only transaction to ensure that the object isn't changed while we're reading it.
- Returns a Student
- Save the Student (and all associated Schedules) in the database
- Transaction Boundaries
- If the Student already has a schedule for that Semester, the existing Schedule is updated with the given Schedule.
Example: Incorporating OODBMS Persistency: Read

1. getStudent(string)

1.1. begin()

1.2. get(string)

1.3. commit(RETAIN_HOLLOW)

Specify the RETAIN_HOLLOW option on the commit(), so that the reference to the retrieved Student can be used outside of the transaction.

Retrieve latest version of the Student from the database.

CourseRegDBRoot : Map
CourseRegDBManager : Transaction
RegistrationController : RegistrationController
Example: Incorporating OODBMS Persistency: Delete

Remove the Student from the database. As a result of the composition relationship between Student and Schedule, all Schedules for the Student will also be deleted.

Remove the Student from physical database. This syncs up the cache with the physical database, which is something that you want to happen immediately for a delete.
Example: Incorporating OODBMS Persistency: VOPC

Object Oriented Analysis and Design

CourseRegDBManager

- initialize()
- shutdown()
- newSchedule(forStudent : Student) : Schedule
- delete(theSchedule : Schedule, forStudent : Student)
- update(theSchedule : Schedule, forStudent : Student)
- save(theSchedule : Schedule, forStudent : Student)
- getSchedule(forStudent : Student, forSemester : Semester) : Schedule
- newStudent() : Student
- delete(theStudent : Student)
- update(theStudent : Student)
- save(theStudent : Student)
- getStudent(withID : string) : Student

Database

- open()
- create()
- createRoot(name : string, object : Object)
- close()

Map

- put()
- get()
- remove()
- new()

Session

- create()
- join()
- terminate()

Schedule

- <<entity>>
- +currentSchedule
- +registrant

Student

- <<entity>>
- <<control>>

Transaction

- begin()
- commit(retain : int)

RegistrationController

- <<control>>

Student

- <<entity>>

Schedule

- <<entity>>

CourseRegDBManager

- <<control>>
Example: Incorporating ObjectStore (cont.)

ObjectStore Support

CourseReg
DBManager (from ObjectStore Support)

University Artifacts

com.odi

Registration

Map (from com.odi)
Session (from com.odi)
Transaction (from com.odi)
Database (from com.odi)