IBM Rational Software

Section 1: Course Registration Requirements

Version 2004

Problem Statement

As the head of information systems for Wylie College you are tasked with developing a new student registration system. The college would like a new client-server system to replace its much older system developed around mainframe technology. The new system will allow students to register for courses and view report cards from personal computers attached to the campus LAN. Professors will be able to access the system to sign up to teach courses as well as record grades.

Due to a decrease in federal funding, the college cannot afford to replace the entire system at once. The college will keep the existing course catalog database where all course information is maintained. This database is an Ingres relational database running on a DEC VAX. Fortunately the college has invested in an open SQL interface that allows access to this database from college's Unix servers. The legacy system performance is rather poor, so the new system must ensure that access to the data on the legacy system occurs in a timely manner. The new system will access course information from the legacy database but will not update it. The registrar's office will continue to maintain course information through another system.

At the beginning of each semester, students may request a course catalogue containing a list of course offerings for the semester. Information about each course, such as professor, department, and prerequisites, will be included to help students make informed decisions.

The new system will allow students to select four course offerings for the coming semester. In addition, each student will indicate two alternative choices in case the student cannot be assigned to a primary selection. Course offerings will have a maximum of ten students and a minimum of three students. A course offering with fewer than three students will be canceled. For each semester, there is a period of time that students can change their schedule. Students must be able to access the system during this time to add or drop courses. Once the registration process is completed for a student, the registration system sends information to the billing system so the student can be billed for the semester. If a course fills up during the actual registration process, the student must be notified of the change before submitting the schedule for processing.

At the end of the semester, the student will be able to access the system to view an electronic report card. Since student grades are sensitive information, the system must employ extra security measures to prevent unauthorized access.

Professors must be able to access the on-line system to indicate which courses they will be teaching. They will also need to see which students signed up for their course offerings. In addition, the professors will be able to record the grades for the students in each class.

Glossary

Introduction

This document is used to define terminology specific to the problem domain, explaining terms, which may be unfamiliar to the reader of the use-case descriptions or other project documents. Often, this document can be used as an informal *data dictionary*, capturing data definitions so that use-case descriptions and other project documents can focus on what the system must do with the information.

Definitions

The glossary contains the working definitions for the key concepts in the Course Registration System.

Course

A class offered by the university.

Course Offering

A specific delivery of the course for a specific semester – you could run the same course in parallel sessions in the semester. Includes the days of the week and times it is offered.

Course Catalog

The unabridged catalog of all courses offered by the university.

Faculty

All the professors teaching at the university.

Finance System

The system used for processing billing information.

Grade

The evaluation of a particular student for a particular course offering.

Professor

A person teaching classes at the university.

Report Card

All the grades for all courses taken by a student in a given semester.

Roster

All the students enrolled in a particular course offering.

Student

A person enrolled in classes at the university.

Schedule

The courses a student has selected for the current semester.

Transcript

The history of the grades for all courses, for a particular student sent to the finance system, which in turn bills the students.

Supplementary Specification

Objectives

The purpose of this document is to define requirements of the Course Registration System. This Supplementary Specification lists the requirements that are not readily captured in the use cases of the use-case model. The Supplementary Specifications and the use-case model together capture a complete set of requirements on the system.

Scope

This Supplementary Specification applies to the Course Registration System, which will be developed by the OOAD students.

This specification defines the non-functional requirements of the system; such as reliability, usability, performance, and supportability, as well as functional requirements that are common across a number of use cases. (The functional requirements are defined in the Use Case Specifications.)

References

None.

Functionality

Multiple users must be able to perform their work concurrently.

If a course offering becomes full while a student is building a schedule including that offering, the student must be notified.

Usability

The desktop user-interface shall be Windows 95/98 compliant.

Reliability

The system shall be available 24 hours a day 7 days a week, with no more than 10% down time.

Performance

The system shall support up to 2000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.

The system shall provide access to the legacy course catalog database with no more than a 10 second latency.
Note: Risk-based prototypes have found that the legacy course catalog database cannot meet our performance needs without some creative use of mid-tier processing power

The system must be able to complete 80% of all transactions within 2 minutes.

Supportability

None.

Security

The system must prevent students from changing any schedules other than their own, and professors from modifying assigned course offerings for other professors.

Only Professors can enter grades for students.

Only the Registrar is allowed to change any student information.

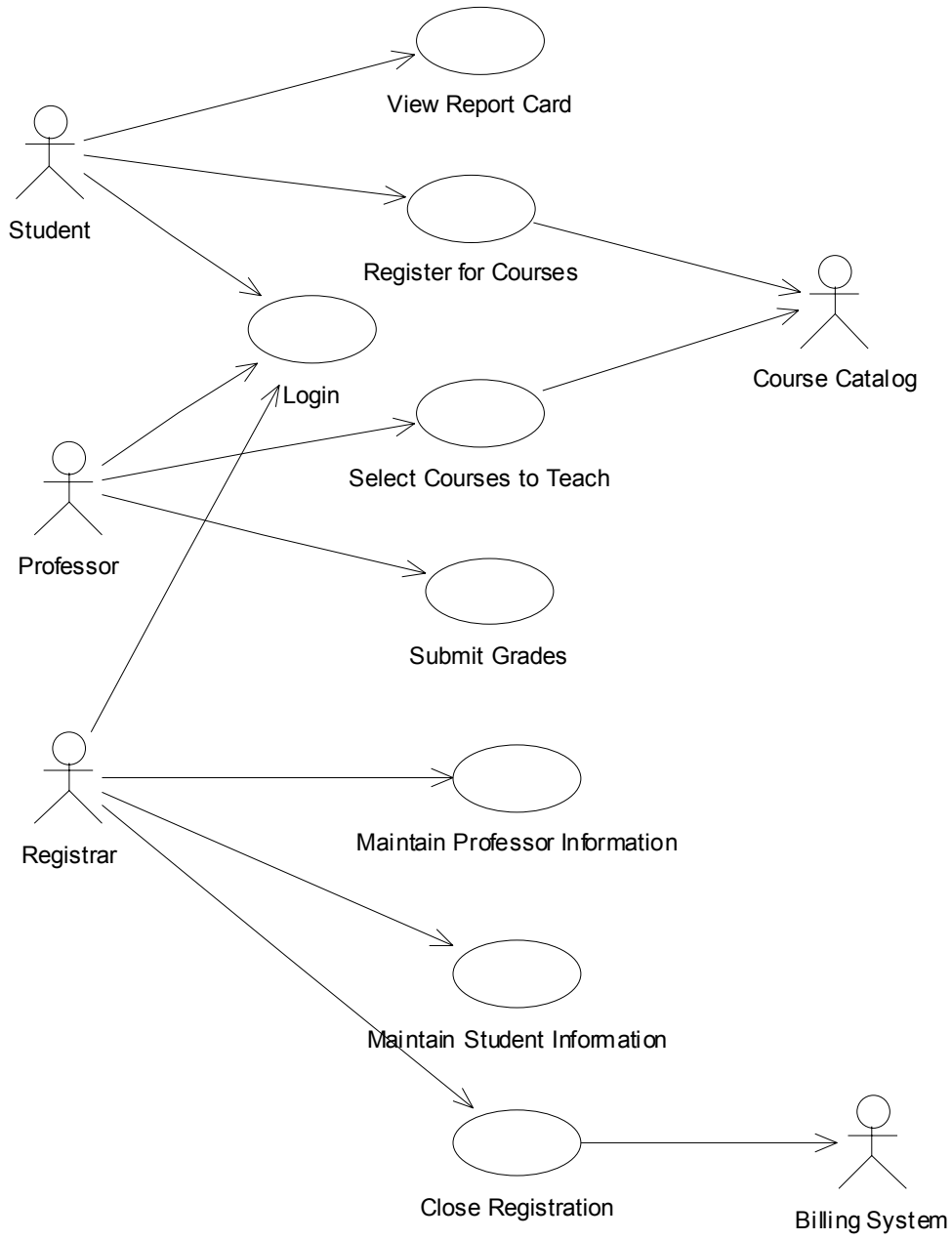
Design Constraints

The system shall integrate with an existing legacy system, the Course Catalog System, which is an RDBMS database.

The system shall provide a Windows-based desktop interface.

Use-Case Model

Course Registration System Use-Case Model Main Diagram



Close Registration

Brief Description

This use case allows a Registrar to close the registration process. Course offerings that do not have enough students are cancelled. Course offerings must have a minimum of three students in them. The billing system is notified for each student in each course offering that is not cancelled, so the student can be billed for the course offering.

Flow of Events

Basic Flow

This use case starts when the Registrar requests that the system close registration.

1. The system checks to see if registration is in progress. If it is, then a message is displayed to the Registrar, and the use case terminates. The Close Registration processing cannot be performed if registration is in progress.
2. For each course offering, the system checks if a professor has signed up to teach the course offering and at least three students have registered. If so, the system commits the course offering for each schedule that contains it.
3. For each schedule, the system "levels" the schedule: if the schedule does not have the maximum number of primary courses selected, the system attempts to select alternates from the schedule's list of alternates. The first available alternate course offerings will be selected. If no alternates are available, then no substitution will be made.
4. For each course offering, the system closes all course offerings. If the course offerings do not have at least three students at this point (some may have been added as a result of leveling), then the system cancels the course offering. The system cancels the course offering for each schedule that contains it.
5. The system calculates the tuition owed by each student for his current semester schedule and sends a transaction to the Billing System. The Billing System will send the bill to the students, which will include a copy of their final schedule.

Alternative Flows

No Professor for the Course Offering

If, in the **Basic Flow**, there is no professor signed up to teach the course offering, the system will cancel the course offering. The system cancels the course offering for each schedule that contains it.

Billing System Unavailable

If the system is unable to communicate with the Billing System, the system will attempt to re-send the request after a specified period. The system will continue to attempt to re-send until the Billing System becomes available.

Special Requirements

None.

Pre-Conditions

The Registrar must be logged onto the system in order for this use case to begin.

Post-Conditions

If the use case was successful, registration is now closed. If not, the system state remains unchanged.

Extension Points

None.

Login

Brief Description

This use case describes how a user logs into the Course Registration System.

Flow of Events

Basic Flow

This use case starts when the actor wishes to log into the Course Registration System.

1. The actor enters his/her name and password.
2. The system validates the entered name and password and logs the actor into the system.

Alternative Flows

Invalid Name/Password

If, in the **Basic Flow**, the actor enters an invalid name and/or password, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends.

Special Requirements

None.

Pre-Conditions

The system is in the login state and has the login screen displayed.

Post-Conditions

If the use case was successful, the actor is now logged into the system. If not, the system state is unchanged.

Extension Points

None.

Maintain Professor Information

Brief Description

This use case allows the Registrar to maintain professor information in the registration system. This includes adding, modifying, and deleting professors from the system.

Flow of Events

Basic Flow

This use case starts when the Registrar wishes to add, change, and/or delete professor information in the system.

1. The system requests that the Registrar specify the function he/she would like to perform (either Add a Professor, Update a Professor, or Delete a Professor)
2. Once the Registrar provides the requested information, one of the sub flows is executed.
   If the Registrar selected "Add a Professor", the **Add a Professor** subflow is executed.
   If the Registrar selected "Update a Professor", the **Update a Professor** subflow is executed.
   If the Registrar selected "Delete a Professor", the **Delete a Professor** subflow is executed.

Add a Professor

The system requests that the Registrar enter the professor information. This includes:
- name
- date of birth
- social security number
- status
- department

1. Once the Registrar provides the requested information, the system generates and assigns a unique id number to the professor. The professor is added to the system.
2. The system provides the Registrar with the new professor id.

Update a Professor

1. The system requests that the Registrar enter the professor id.
2. The Registrar enters the professor id. The system retrieves and displays the professor information.
3. The Registrar makes the desired changes to the professor information. This includes any of the information specified in the **Add a Professor** sub-flow.
4. Once the Registrar updates the necessary information, the system updates the professor record.

Delete a Professor

1. The system requests that the Registrar enter the professor id
2. The Registrar enters the professor id. The system retrieves and displays the professor information.
3. The system prompts the Registrar to confirm the deletion of the professor.
4. The Registrar verifies the deletion.
5. The system deletes the professor from the system.

Alternative Flows

Professor Not Found

If, in the **Update a Professor** or **Delete a Professor** sub-flows, a professor with the specified id number does not exist, the system displays an error message. The Registrar can then enter a different id number or cancel the operation, at which point the use case ends.

Delete Cancelled

If, in the **Delete A Professor** sub-flow, the Registrar decides not to delete the professor, the delete is cancelled, and the **Basic Flow** is re-started at the beginning.

Special Requirements

None.

Pre-Conditions

The Registrar must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the professor information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

Extension Points

None.

Maintain Student Information

Brief Description

This use case allows the Registrar to maintain student information in the registration system. This includes adding, modifying, and deleting Students from the system.

Flow of Events

Basic Flow

This use case starts when the Registrar wishes to add, change, and/or delete student information in the system.

1. The system requests that the Registrar specify the function he/she would like to perform (either Add a Student, Update a Student, or Delete a Student)
2. Once the Registrar provides the requested information, one of the sub flows is executed.
   If the Registrar selected "Add a Student", the **Add a Student** subflow is executed.
   If the Registrar selected "Update a Student", the **Update a Student** subflow is executed.
   If the Registrar selected "Delete a Student", the **Delete a Student** subflow is executed.

Add a Student

1. The system requests that the Registrar enter the student information. This includes:
   - name
   - date of birth
   - social security number
   - status
   - graduation date
2. Once the Registrar provides the requested information, the system generates and assigns a unique id number to the student. The student is added to the system.
3. The system provides the Registrar with the new student id.

Update a Student

1. The system requests that the Registrar enter the student id.
2. The Registrar enters the student id. The system retrieves and displays the student information.
3. The Registrar makes the desired changes to the student information. This includes any of the information specified in the **Add a Student** sub-flow.
4. Once the Registrar updates the necessary information, the system updates the student information.

Delete a Student

1. The system requests that the Registrar enter the student id
2. The Registrar enters the student id. The system retrieves and displays the student information.
3. The system prompts the Registrar to confirm the deletion of the student.
4. The Registrar verifies the deletion.
5. The system deletes the student from the system.

Alternative Flows

Student Not Found

If, in the **Update a Student** or **Delete a Student** sub-flows, a student with the specified id number does not exist, the system displays an error message. The Registrar can then enter a different id number or cancel the operation, at which point the use case ends.

Delete Cancelled

If, in the **Delete A Student** sub-flow, the Registrar decides not to delete the student, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

Special Requirements

None.

Pre-Conditions

The Registrar must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the student information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

Extension Points

None.

Register for Courses

Brief Description

This use case allows a Student to register for course offerings in the current semester. The Student can also update or delete course selections if changes are made within the add/drop period at the beginning of the semester. The Course Catalog System provides a list of all the course offerings for the current semester.

Flow of Events

Basic Flow

This use case starts when a Student wishes to register for course offerings, or to change his/her existing course schedule.

1. The Student provides the function to perform (one of the sub flows is executed):
   If the Student selected "Create a Schedule", the **Create a Schedule** subflow is executed.
   If the Student selected "Update a Schedule", the **Update a Schedule** subflow is executed.
   If the Student selected "Delete a Schedule", the **Delete a Schedule** subflow is executed.

Create a Schedule

1. The system retrieves a list of available course offerings from the Course Catalog System and displays the list to the Student.
2. The Select Offerings subflow is executed.
3. The Submit Schedule subflow is executed.

Update a Schedule

1. The system retrieves and displays the Student's current schedule (e.g., the schedule for the current semester).
2. The system retrieves a list of available course offerings from the Course Catalog System and displays the list to the Student.
3. The Student may update the course selections on the current selection by deleting and adding new course offerings. The Student selects the course offerings to add from the list of available course offerings. The Student also selects any course offerings to delete from the existing schedule.
4. Once the student has made his/her selections, the system updates the schedule for the Student using the selected course offerings.
5. The Submit Schedule subflow is executed.

Delete a Schedule

1. The system retrieves and displays the Student's current schedule (e.g., the schedule for the current semester).
2. The system prompts the Student to confirm the deletion of the schedule.
3. The Student verifies the deletion.
4. The system deletes the Schedule. If the schedule contains "enrolled in" course offerings, the Student must be removed from the course offering.

Select Offerings

The Student selects 4 primary course offerings and 2 alternate course offerings from the list of available offerings.

Once the student has made his/her selections, the system creates a schedule for the Student containing the selected course offerings.

Submit Schedule

For each selected course offering on the schedule not already marked as "enrolled in", the system verifies that the Student has the necessary prerequisites, that the course offering is open, and that there are no schedule conflicts.

The system then adds the Student to the selected course offering. The course offering is marked as "enrolled in" in the schedule.

The schedule is saved in the system.

Alternative Flows

Save a Schedule

At any point, the Student may choose to save a schedule rather than submitting it. If this occurs, the Submit Schedule step is replaced with the following:

The course offerings not marked as "enrolled in" are marked as "selected" in the schedule.

The schedule is saved in the system.

Unfulfilled Prerequisites, Course Full, or Schedule Conflicts

If, in the **Submit Schedule** sub-flow, the system determines that the Student has not satisfied the necessary prerequisites, or that the selected course offering is full, or that there are schedule conflicts, an error message is displayed. The Student can either select a different course offering and the use case continues, save the schedule, as is (see **Save a Schedule** subflow), or cancel the operation, at which point the **Basic Flow** is re-started at the beginning.

No Schedule Found

If, in the **Update a Schedule** or **Delete a Schedule** sub-flows, the system is unable to retrieve the Student's schedule, an error message is displayed. The Student acknowledges the error, and the **Basic Flow** is re-started at the beginning.

Course Catalog System Unavailable

If the system is unable to communicate with the Course Catalog System, the system will display an error message to the Student. The Student acknowledges the error message, and the use case terminates.

Course Registration Closed

When the use case starts, if it is determined that registration for the current semester has been closed, a message is displayed to the Student, and the use case terminates. Students cannot register for course offerings after registration for the current semester has been closed.

Delete Cancelled

If, in the **Delete A Schedule** sub-flow, the Student decides not to delete the schedule, the delete is cancelled, and the **Basic Flow** is re-started at the beginning.

Special Requirements

None.

Pre-Conditions

The Student must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the student schedule is created, updated, or deleted. Otherwise, the system state is unchanged.

Extension Points

None.

Select Courses to Teach

Brief Description

This use case allows a Professor to select the course offerings from the course catalog for the courses that he/she is eligible for and wishes to teach in the upcoming semester.

Flow of Events

Basic Flow

This use case starts when a Professor wishes to sign up to teach some course offerings for the upcoming semester.

1. The system retrieves and displays the list of course offerings the professor is eligible to teach for the current semester. The system also retrieves and displays the list of courses the professor has previously selected to teach.
2. The professor selects and/or de-selects the course offerings that he/she wishes to teach for the upcoming semester.
3. The system removes the professor from teaching the de-selected course offerings.
4. The system verifies that the selected offerings do not conflict (i.e., have the same dates and times) with each other or any course offerings that the professor has previously signed up to teach. If there is no conflict, the system updates the course offering information for each offering the professor selects (i.e., records the professor as the instructor for the course offering).

Alternative Flows

No Course Offerings Available

If, in the **Basic Flow**, the professor is not eligible to teach any course offerings in the upcoming semester, the system will display an error message. The professor acknowledges the message and the use case ends.

Schedule Conflict

If the systems find a schedule conflict when trying to establish the course offerings the Professor should take, the system will display an error message indicating that a schedule conflict has occurred. The system will also indicate which are the conflicting courses. The Professor can either resolve the schedule conflict (i.e., by canceling his selection to teach one of the course offerings), or cancel the operation, in which case, any selections will be lost, and the use case ends.

Course Catalog System Unavailable

If the system is unable to communicate with the Course Catalog System, the system will display an error message to the Student. The Student acknowledges the error message, and the use case terminates.

Course Registration Closed

When the use case starts, if it is determined that registration for the current semester has been closed, a message is displayed to the Professor, and the use case terminates. Professors cannot change the course offerings they teach after registration for the current semester has been closed. If a professor change is needed after registration has been closed, it is handled outside the scope of this system.

Special Requirements

None.

Pre-Conditions

The Professor must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the course offerings a Professor is scheduled to teach have been updated. Otherwise, the system state is unchanged.

Extension Points

None.

Submit Grades

Brief Description

This use case allows a Professor to submit student grades for one or more classes completed in the previous semester.

Flow of Events

Basic Flow

This use case starts when a Professor wishes to submit student grades for one or more classes completed in the previous semester.

1. The system displays a list of course offerings the Professor taught in the previous semester.
2. The Professor selects a course offering.
3. The system retrieves a list of all students who were registered for the course offering. The system displays each student and any grade that was previously assigned for the offering.
4. For each student on the list, the Professor enters a grade: A, B, C, D, F, or I. The system records the student's grade for the course offering. If the Professor wishes to skip a particular student, the grade information can be left blank and filled in at a later time. The Professor may also change the grade for a student by entering a new grade.

Alternative Flows

No Course Offerings Taught

If, in the **Basic Flow**, the Professor did not teach any course offerings in the previous semester, the system will display an error message. The Professor acknowledges the message, and the use case ends.

Special Requirements

None.

Pre-Conditions

The Professor must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, student grades for a course offering are updated. Otherwise, the system state is unchanged.

Extension Points

None.

View Report Card

Brief Description

This use case allows a Student to view his/her report card for the previously completed semester.

Flow of Events

Basic Flow

This use case starts when a Student wishes to view his/her report card for the previously completed semester.

1. The system retrieves and displays the grade information for each of the course offerings the Student completed during the previous semester.
2. When the Student indicates that he/she is done viewing the grades, the use case terminates.

Alternative Flows

No Grade Information Available

If, in the **Basic Flow**, the system cannot find any grade information from the previous semester for the Student, a message is displayed. Once the Student acknowledges the message, the use case terminates.

Special Requirements

None.

Pre-Conditions

The Student must be logged onto the system before this use case begins.

Post-Conditions

The system state is unchanged by this use case.

Extension Points

None.

Section 2: Payroll Requirements

Version 2004

Problem Statement

As the head of Information Technology at Acme, Inc., you are tasked with building a new payroll system to replace the existing system, which is hopelessly out of date. Acme needs a new system to allow employees to record timecard information electronically and automatically generate paychecks based on the number of hours worked and total amount of sales (for commissioned employees).

The new system will be state of the art and will have a Windows-based desktop interface to allow employees to enter timecard information, enter purchase orders, change employee preferences (such as payment method), and create various reports. The system will run on individual employee desktops throughout the entire company. For reasons of security and auditing, employees can only access and edit their own timecards and purchase orders.

The system will retain information on all employees in the company (Acme currently has around 5,000 employees world-wide). The system must pay each employee the correct amount, on time, by the method that they specify (see possible payment methods described later). Acme, for cost reasons, does not want to replace one of their legacy databases, the Project Management Database, which contains all information regarding projects and charge numbers. The new system must work with the existing Project Management Database, which is a DB2 database running on an IBM mainframe. The Payroll System will access, but not update, information stored in the Project Management Database.

Some employees work by the hour, and they are paid an hourly rate. They submit timecards that record the date and number of hours worked for a particular charge number. If someone works for more than 8 hours, Acme pays them 1.5 times their normal rate for those extra hours. Hourly workers are paid every Friday.

Some employees are paid a flat salary. Even though they are paid a flat salary, they submit timecards that record the date and hours worked. This is so the system can keep track of the hours worked against particular charge numbers. They are paid on the last working day of the month.

Some of the salaried employees also receive a commission based on their sales. They submit purchase orders that reflect the date and amount of the sale. The commission rate is determined for each employee, and is one of 10%, 15%, 25%, or 35%.

One of the most requested features of the new system is employee reporting. Employees will be able to query the system for number of hours worked, totals of all hours billed to a project (i.e., charge number), total pay received year-to-date, remaining vacation time, etc.

Employees can choose their method of payment. They can have their paychecks mailed to the postal address of their choice, or they can request direct deposit and have their paycheck deposited into a bank account of their choosing. The employee may also choose to pick their paychecks up at the office.

The Payroll Administrator maintains employee information. The Payroll Administrator is responsible for adding new employees, deleting employees and changing all employee information such as name, address, and payment classification (hourly, salaried, commissioned), as well as running administrative reports.

The payroll application will run automatically every Friday and on the last working day of the month. It will pay the appropriate employees on those days. The system will be told what date the employees are to be paid, so it will generate payments for records from the last time the employee was paid to the specified date. The new system is being designed so that the payroll will always be generated automatically, and there will be no need for any manual intervention.

Glossary

Introduction

This document is used to define terminology specific to the problem domain, explaining terms, which may be unfamiliar to the reader of the use-case descriptions or other project documents. Often, this document can be used as an informal *data dictionary*, capturing data definitions so that use-case descriptions and other project documents can focus on what the system must do with the information.

Definitions

The glossary contains the working definitions for the key concepts in the Payroll System.

Bank System

Any bank(s) to which direct deposit transactions are sent.

Employee

A person that works for the company that owns and operates the payroll system (Acme, Inc.)

Payroll Administrator

The person responsible for maintaining employees and employee information in the system.

Project Management Database

The legacy database that contains all information regarding projects and charge numbers.

System Clock

The internal system clock that keeps track of time. The internal clock will automatically run the payroll at the appropriate times.

Pay Period

The amount of time over which an employee is paid.

Paycheck

A record of how much an employee was paid during a specified Pay Period.

Payment Method

How the employee is paid, either pick-up, mail, or direct deposit.

Timecard

A record of hours worked by the employee during a specified pay period.

Purchase Order

A record of a sale made by an employee.

Salaried Employee

An employee that receives a salary.

Commissioned Employee

An employee that receives a salary plus commissions.

Hourly Employee

An employee that is paid by the hour.

Supplementary Specification

Objectives

The purpose of this document is to define requirements of the Payroll System. This Supplementary Specification lists the requirements that are not readily captured in the use cases of the use-case model. The Supplementary Specifications and the use-case model together capture a complete set of requirements on the system.

Scope

This Supplementary Specification applies to the Payroll System, which will be developed by the OOAD students.

This specification defines the non-functional requirements of the system; such as reliability, usability, performance, and supportability as well as functional requirements that are common across a number of use cases. (The functional requirements are defined in the Use Case Specifications.).

References

None.

Functionality

None.

Usability

None.

Reliability

The main system must be running 98% of the time. It is imperative that the system be up and running during the times the payroll is run (every Friday and the last working day of the month).

Performance

The system shall support up to 2000 simultaneous users against the central database at any given time, and up to 500 simultaneous users against the local servers at any one time.

Supportability

None.

Security

The system should prevent employees from changing any timecards other than their own. Additionally, for security reasons, only the Payroll Administrator is allowed to change any employee information with the exception of the payment delivery method.

Design Constraints

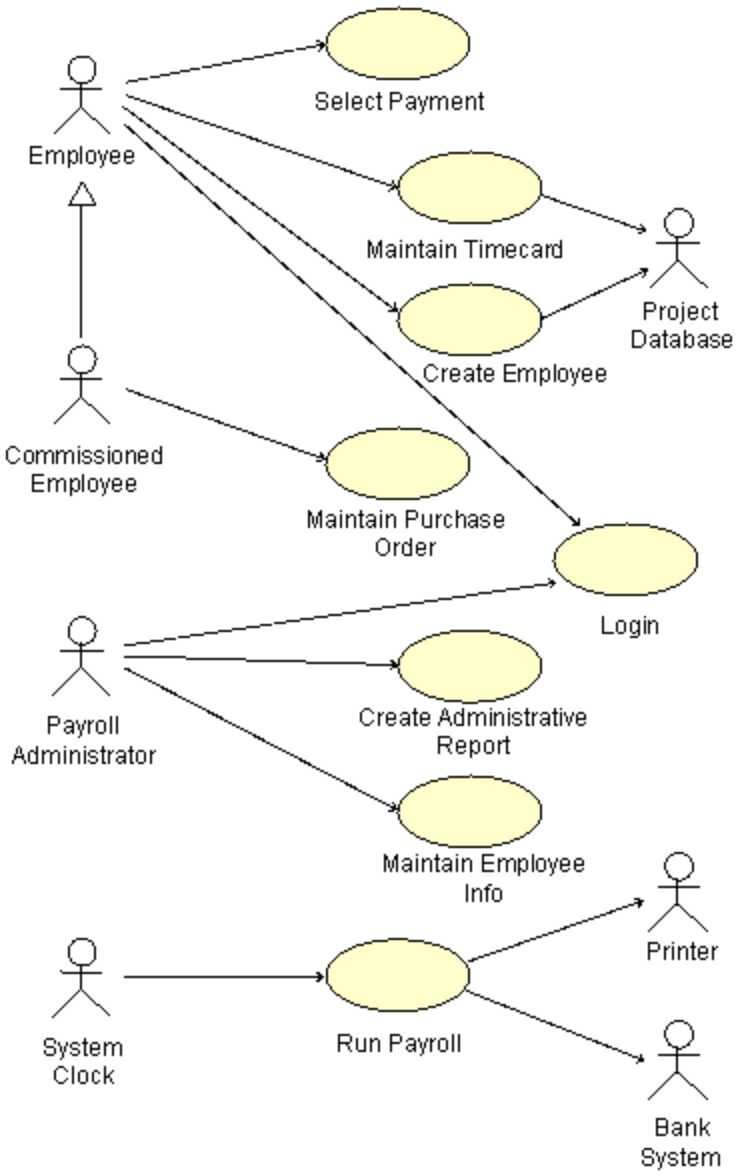
The system shall integrate with an existing legacy system, the Project Management Database, which is a DB2 database running on an IBM mainframe.

The system shall interface with existing bank systems via an electronic transaction interface (NOTE: THE FORMAL INTERFACES WITH THE EXTERNAL BANK SYSTEM WOULD NEED TO BE DEFINED EARLY IN THE PROCESS AND DEFINED HERE OR IN A SEPARATE SUPPORTING DOCUMENT. SUCH A DEFINITION IS OUT OF THE SCOPE OF THIS COURSE.)

The system shall provide a Windows-based desktop interface.

Use-Case Model

Payroll System Use-Case Model Main Diagram



Create Administrative Report

Brief Description

The use case allows the Payroll Administrator to create either a “Total Hours Worked” or “Pay Year-to-Date” report.

Flow of Events

Basic Flow

The use case begins when the Payroll Administrator requests that the system create an administrative report.

1. The system requests that the Payroll Administrator specify the following report criteria:

   - Report Type (either total hours worked or pay year-to-date),
   - Begin and end dates for the report,
   - Employee name(s)

2. Once the Payroll Administrator provides the requested information, the system provides the Payroll Administrator with a report satisfying the report criteria.
3. The Payroll Administrator may then request that the system save the report. At which time, the system requests the Payroll Administrator to provide the name and location for saving the report.
4. Once the Payroll Administrator provides the requested information and confirms the decision to save the report, the system saves the report to the specified name and location.
5. If the Payroll Administrator did not elect to save the report, the report is discarded.

Alternative Flows

Requested Information Unavailable

If in the **Basic Flow**, the requested information is unavailable, the system will display an error message. The Payroll Administrator can choose to either return to the beginning of the **Basic Flow,** or cancel the operation, at which point the use case ends.

Invalid Format or Insufficient Information

If, in the **Basic Flow**, the Payroll Administrator has not specified sufficient information to create the selected report, the system will prompt the actor for the missing information. The Payroll Administrator can either enter the missing information or choose to cancel the operation, at which point the use case ends.

Special Requirements

None.

Pre-Conditions

The Payroll Administrator must be logged onto the system in order for this use case to begin.

Post-Conditions

The system state is unchanged by this use case.

Extension Points

None.

Create Employee Report

Brief Description

The use case allows the Employee to create a "Total Hours Worked," "Total Hours Worked for a Project", "Vacation/Sick Leave," or "Total Pay Year-to-Date" report.

Flow of Events

Basic Flow

This use case starts when the Employee wishes to create a "Total Hours Worked," "Total Hours Worked for a Project", "Vacation/Sick Leave," or "Total Pay Year-to-Date" report.

1. The system requests that the Employee specify the following report criteria:

   - Report Type (either "Total Hours Worked," "Total Hours Worked for a Project", "Vacation/Sick Leave," or "Total Pay Year-to-Date")
   - Begin and end dates for the report

2. If the Employee selected the "Total Hours Worked for a Project" report, the system retrieves and displays a list of the available charge numbers from the Project Management Database. The system then requests that the Employee select a charge number.
3. Once the Employee provides the requested information, the system provides the Employee with a report satisfying the report criteria.
4. The Employee may then request that the system save the report. At which time, the system requests the Employee to provide the name and location for saving the report.
5. Once the Employee provides the requested information and confirms the decision to save the report, the system saves the report to the specified name and location.
6. If the Employee did not elect to save the report, the report is discarded.

Alternative Flows

Requested Information Unavailable

If, in the **Basic Flow**, the requested information is unavailable, the system will display an error message. The Employee can choose to either return to the beginning of the **Basic Flow,** or cancel the operation, at which point the use case ends.

Invalid Format or Insufficient Information

If, in the **Basic Flow**, the Employee has not specified sufficient information to create the selected report, the system will prompt the actor for the missing information. The Employee can either enter the missing information or choose to cancel the operation, at which point the use case ends.

Special Requirements

None.

Pre-Conditions

The Employee must be logged onto the system before this use case begins.

Post-Conditions

The system state is unchanged by this use case.

Extension Points

None.

Login

Brief Description

This use case describes how a user logs into the Payroll System.

Flow of Events

Basic Flow

This use case starts when the actor wishes to Login to the Payroll System.

1. The actor enters his/her name and password.
2. The system validates the entered name and password and logs the actor into the system.

Alternative Flows

Invalid Name/Password

If, in the **Basic Flow**, the actor enters an invalid name and/or password, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends.

Special Requirements

None.

Pre-Conditions

The system is in the login state and has the login screen displayed.

Post-Conditions

If the use case was successful, the actor is now logged into the system. If not, the system state is unchanged.

Extension Points

None.

Maintain Employee Information

Brief Description

This use case allows the Payroll Administrator to maintain employee information. This includes adding, changing, and deleting employee information from the system.

Flow of Events

Basic Flow

This use case starts when the Payroll Administrator wishes to add, change, and/or delete employee information from the system.

1. The system requests that the Payroll Administrator specify the function he/she would like to perform (either Add an Employee, Update an Employee, or Delete an Employee)
2. Once the Payroll Administrator provides the requested information, one of the sub flows is executed.
   If the Payroll Administrator selected "Add an Employee", the **Add an Employee** subflow is executed.
   If the Payroll Administrator selected "Update an Employee", the **Update an Employee** subflow is executed.
   If the Payroll Administrator selected "Delete an Employee", the **Delete an Employee** subflow is executed.

Add an Employee

1. The system requests that the Payroll Administrator enter the employee information. This includes:
2. name
   - employee type (hour, salaried, commissioned)
   - mailing address
   - social security number
   - standard tax deductions
   - other deductions (401k, medical)
   - phone number
   - hourly rate (for hourly employees)
   - salary (for salaried and commissioned employees)
   - commission rate (for commissioned employees)
   - hour limit (some employees may not be able to work overtime)
3. Once the Payroll Administrator provides the requested information, the system generates and assigns a unique employee id number to the employee and sets the paycheck delivery method to default of "pickup". The employee is added to the system.
4. The system provides the Payroll Administrator with the new employee id.

Update an Employee

1. The system requests that the Payroll Administrator enter the employee id.
2. The Payroll Administrator enters the employee id. The system retrieves and displays the employee information.
3. The Payroll Administrator makes the desired changes to the employee information. This includes any of the information specified in the Add an Employee sub-flow.
4. Once the Payroll Administrator updates the necessary information, the system updates the employee record with the updated information.

Delete an Employee

1. The system requests that the Payroll Administrator specify the employee id.

2. The Payroll Administrator enters the employee id. The system retrieves and displays the employee information.
3. The system prompts the Payroll Administrator to confirm the deletion of the employee.
4. The Payroll Administrator verifies the deletion.
5. The system marks the employee record for deletion. The next time the payroll is run, the system will generate a final paycheck for the deleted employee and remove the employee from the system.

Alternative Flows

Employee Not Found

If in the **Update an Employee or Delete an Employee** sub-flows, an employee with the specified id number does not exist, the system displays an error message. The Payroll Administrator can then enter a different id number or cancel the operation, at which point the use case ends.

Delete Cancelled

If in the **Delete An Employee** sub-flow, the Payroll Administrator decides not to delete the employee, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

Special Requirements

None.

Pre-Conditions

The Payroll Administrator must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the employee information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

Extension Points

None.

Maintain Purchase Order

Brief Description

This use case allows a Commissioned Employee to record and maintain purchase orders. This includes adding, changing, and deleting purchase orders. Commissioned employees must record each of their purchase orders in order to receive commissions.

Flow of Events

Basic Flow

This use case starts when the Commissioned Employee wishes to add, change, and/or delete purchase order information from the system.

1. The system requests that the Commissioned Employee specify the function he/she would like to perform (either Create a Purchase Order, Update a Purchase Order, or Delete a Purchase Order)
2. Once the Commissioned Employee provides the requested information, one of the sub flows is executed.
   If the Commissioned Employee selected "Create a Purchase Order ", the **Create a Purchase Order** subflow is executed.
   If the Commissioned Employee selected "Update a Purchase Order ", the **Update a Purchase Order** subflow is executed.
   If the Commissioned Employee selected "Delete a Purchase Order ", the **Delete a Purchase Order** subflow is executed.

Create a Purchase Order

1. The system requests that the Commissioned Employee enter the purchase order information. This includes:

   - customer point of contact
   - customer billing address
   - product(s) purchased
   - date

2. Once the Commissioned Employee provides the requested information, the system generates and assigns a unique purchase order number to the purchase order. The purchase order is added to the system for the Commissioned Employee.
3. The system provides the Commissioned Employee with the new purchase order id.

Update a Purchase Order

1. The system requests that the Commissioned Employee enter the purchase order id.
2. The Commissioned Employee enters the purchase order id.
3. The system retrieves the purchase order associated with the purchase order id.
4. The system verifies that the purchase order is a purchase order for the Commissioned Employee, and that the purchase order is open.
5. The system displays the purchase order.
6. The Commissioned Employee makes the desired changes to the purchase order information. This includes any of the information specified in the **Create a Purchase Order** sub flow.
7. Once the Commissioned Employee updates the necessary information, the system updates the purchase order with the updated information.

Delete a Purchase Order

1. The system requests that the Commissioned Employee specify the purchase order id.
2. The Commissioned Employee enters the purchase order id.
3. The system retrieves the purchase order associated with the purchase order id.
4. The system verifies that the purchase order is a purchase order for the Commissioned Employee, and that the purchase order is open.
5. The system displays the purchase order.
6. The system prompts the Commissioned Employee to confirm the deletion of the purchase order.
7. The Commissioned Employee verifies the deletion.
8. The system removes the purchase order from the system.

Alternative Flows

Purchase Order Not Found

If, in the **Update a Purchase Order or Delete an Purchase Order** sub-flows, an purchase order with the specified id number does not exist, the system displays an error message. The Commissioned Employee can then enter a different id number or cancel the operation, at which point the use case ends.

Invalid Access to a Purchase Order

If, in the **Update a Purchase Order or Delete a Purchase Order** sub-flows, the Commissioned Employee attempts to access a purchase order that is not his, the system displays an error message. The Commissioned Employee can then enter a different id number or cancel the operation, at which point the use case ends.

Purchase Order is Closed

If, in the **Update a Purchase Order or Delete a Purchase Order** sub-flows, the Commissioned Employee attempts to access a purchase order that is closed, the system displays an error message. The Commissioned Employee can then enter a different id number or cancel the operation, at which point the use case ends.

Delete Cancelled

If, in the **Delete A Purchase Order** sub-flow, the Commissioned Employee decides not to delete the purchase order, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

Special Requirements

None.

Pre-Conditions

The Commissioned Employee must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the purchase order information is added, updated, or deleted from the system. Otherwise, the system state is unchanged.

Extension Points

None.

Maintain Timecard

Brief Description

This use case allows the Employee to update and submit timecard information. Hourly and salaried employees must submit weekly timecards recording all hours worked that week and which projects the hours are billed to. An Employee can only make changes to the timecard for the current pay period and before the timecard has been submitted.

Flow of Events

Basic Flow

This use case starts when the Employee wishes to enter hours worked into his current timecard.

1. The system retrieves and displays the current timecard for the Employee. If a timecard does not exist for the Employee for the current pay period, the system creates a new one. The start and end dates of the timecard are set by the system and cannot be changed by the Employee.
2. The system retrieves and displays the list of available charge numbers from the Project Management Database.
3. The Employee selects the appropriate charge numbers and enters the hours worked for any desired date (within the date range of the timecard).
4. Once the Employee has entered the information, the system saves the timecard.

Submit Timecard

1. At any time, the Employee may request that the system submit the timecard.
2. At that time, the system assigns the current date to the timecard as the submitted date and changes the status of the timecard to "submitted." No changes are permitted to the timecard once it has been submitted.
3. The system validates the timecard by checking the number of hours worked against each charge number. The total number of hours worked against all charge numbers must not exceed any limit established for the Employee (for example, the Employee may not be allowed to work overtime).
4. The system retains the number of hours worked for each charge number in the timecard.
5. The system saves the timecard.
6. The system makes the timecard read-only, and no further changes are allowed once the timecard is submitted.

Alternative Flows

Invalid Number of Hours

If, in the **Basic Flow,** an invalid number of hours is entered for a single day (>24), or the number entered exceeds the maximum allowable for the Employee, the system will display an error message and prompt for a valid number of hours. The Employee must enter a valid number, or cancel the operation, in which case the use case ends.

Timecard Already Submitted

If, in the **Basic Flow,** the Employee's current timecard has already been submitted, the system displays a read-only copy of the timecard and informs the Employee that the timecard has already been submitted, so no changes can be made to it. The Employee acknowledges the message and the use case ends.

Project Management Database Not Available

If, in the **Basic Flow,** the Project Management Database is not available, the system will display an error message stating that the list of available charge numbers is not available. The Employee acknowledges the error and may either choose to continue (without selectable charge numbers), or to cancel (any timecard changes are discarded and the use case ends).
Note: Without selectable charge numbers, the Employee may change hours for a charge number already listed on the timecard, but he/she may not add hours for a charge number that is not already listed.

Special Requirements

None.

Pre-Conditions

The Employee must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the Employee timecard information is saved to the system. Otherwise, the system state is unchanged.

Extension Points

None.

Run Payroll

Brief Description

The use case describes how the payroll is run every Friday and the last working day of the month.

Flow of Events

Basic Flow

1. The use case begins when it's time to run the payroll. The payroll is run automatically every Friday and the last working day of the month.
2. The system retrieves all employees who should be paid on the current date.
3. The system calculates the pay using entered timecards, purchase orders, employee information (e.g., salary, benefits, etc.) and all legal deductions.
4. If the payment delivery method is mail or pick-up, the system prints a paycheck.
5. If the payment delivery method is direct deposit, the system creates a bank transaction and sends it to the Bank System for processing.
6. The use case ends when all employees receiving pay for the desired date have been processed.

Alternative Flows

Bank System Unavailable

If the Bank System is down, the system will attempt to send the bank transaction again after a specified period. The system will continue to attempt to re-transmit until the Bank System becomes available.

Deleted Employees

After the payroll for an Employee has been processed, if the employee has been marked for deletion (see the **Maintain Employee** use case), then the system will delete the employee.

Special Requirements

None.

Pre-Conditions

None.

Post-Conditions

Payments for each employee eligible to be paid on the current date have been processed.

Extension Points

None.

Select Payment Method

Brief Description

This use case allows an Employee to select a payment method. The payment method controls how the Employee will be paid. The Employee may choose to either: pick up his check directly, receive it in the mail, or have it deposited directly into a specified bank account.

Flow of Events

Basic Flow

This use case starts when the Employee wishes to select a payment method.

1. The system requests that the Employee specify the payment method he would like (either: “pick up”, “mail”, or “direct deposit”).
2. The Employee selects the desired payment method.
3. If the Employee selects the “pick-up” payment method, no additional information is required.
   If the Employee selects the “mail” payment method, the system requests that the Employee specify the address that the paycheck will be mailed to.
   If the Employee selects the “direct deposit” method, the system requests that the Employee specify the bank name and account number.
4. Once the Employee provides the requested information, the system updates the Employee information to reflect the chosen payment method.

Alternative Flows

Employee Not Found

If, in the **Basic Flow**, information for the employee could not be located, the system displays an error message, and the use case ends.

Special Requirements

None.

Pre-Conditions

The Employee must be logged onto the system before this use case begins.

Post-Conditions

If the use case was successful, the payment method for the Employee is updated in the system. Otherwise, the system state is unchanged.

Extension Points

None.

Section 3: Payroll Architecture Handbook

Version 2004

Payroll Architecture Handbook

Description

This document supplements the course material for the Payroll Exercise used in the Object-Oriented Analysis and Design Using the UML course. It provides the architectural givens that support the development of the Payroll System design model during the course exercises.
This is because the OOAD course concentrates on demonstrating how architecture affects the design model. OOAD is NOT an architecture course. The OOAD course gives the students an appreciation of what an architecture is and why it is important.

In some sections of this document, the architecture is represented textually. The students, as part of the exercises throughout the course, will generate the associated UML diagrams. Thus, for the UML representation of the architecture, see the Payroll Exercise Solution.

Note: A SUBSET OF THE PAYROLL SYSTEM IS PROVIDED. Concentration is on the elements needed to support the Login, Maintain Timecard and Run Payroll use cases.

Architectural Mechanisms

Analysis Mechanisms

Persistency: A means to make an element persistent (i.e., exist after the application that created it ceases to exist).
Distribution: A means to distribute an element across existing nodes of the system.
Note: For this course, it has been decided that the business logic will be distributed.
Security: A means to control access to an element.
Legacy Interface: A means to access a legacy system with an existing interface.

Analysis-to-Design-to-Implementation Mechanisms Map

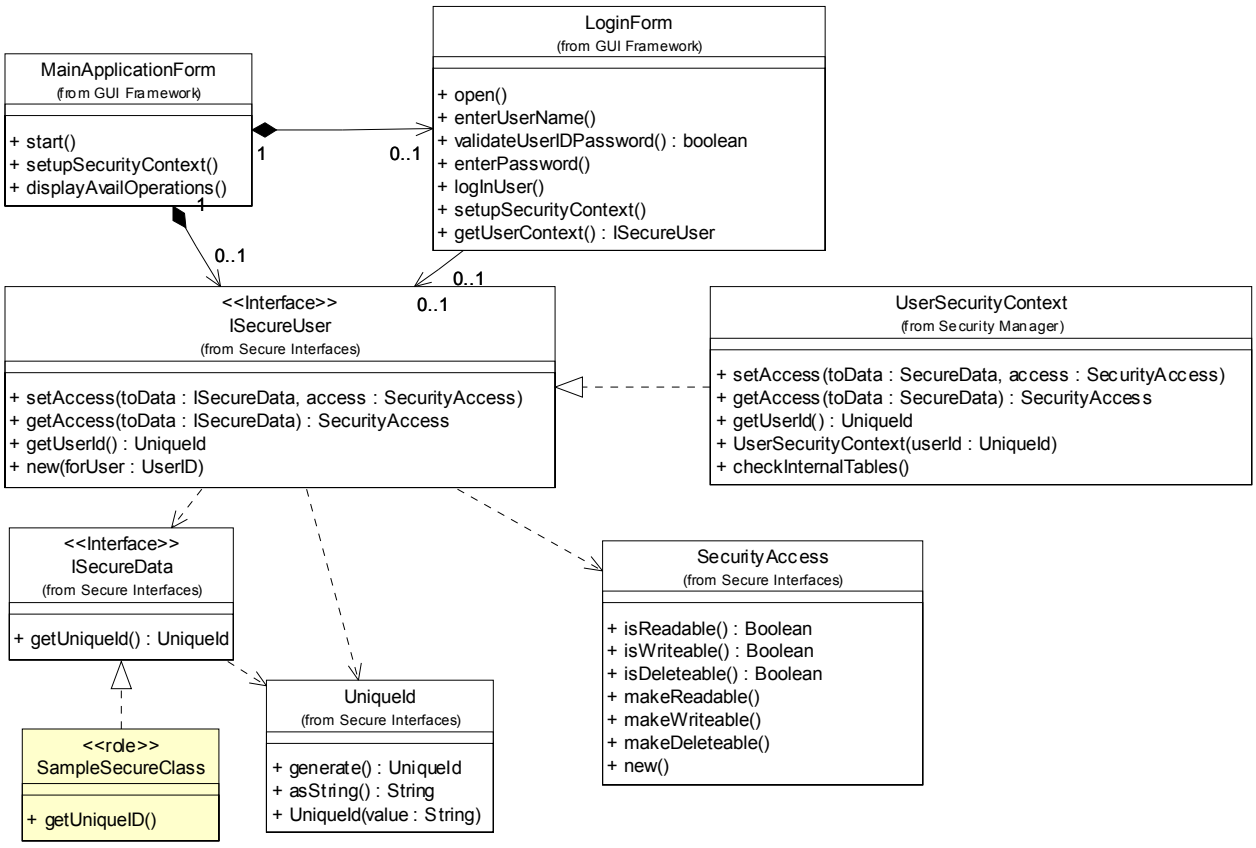
Analysis Mechanism	Design Mechanisms	Implementation Mechanisms
Persistency	OODBMS (new data)	ObjectStore
Persistency	RDBMS (data from legacy database)	JDBC to Ingres
Distribution	Remote Method Invocation (RMI)	Java 1.1 from Sun
Security		Reverse Engineered Secure.java and UserContextRemoteObject components
Legacy Interface		

Note: Remote Method Invocation (RMI) is a Java-specific mechanism that allows client objects to invoke operations on server objects as though they were local. Native Java RMI comes with Sun's Java 1.1.

Implementation Mechanisms

Security

Static View: Security



Class Descriptions

ISecureData : Analysis Mechanisms:
- Security

SecurityAccess : Analysis Mechanisms:
- Security

SampleSecureClass :

UserSecurityContext : Analysis Mechanisms:
- Security

UniqueId : Analysis Mechanisms:
- Security

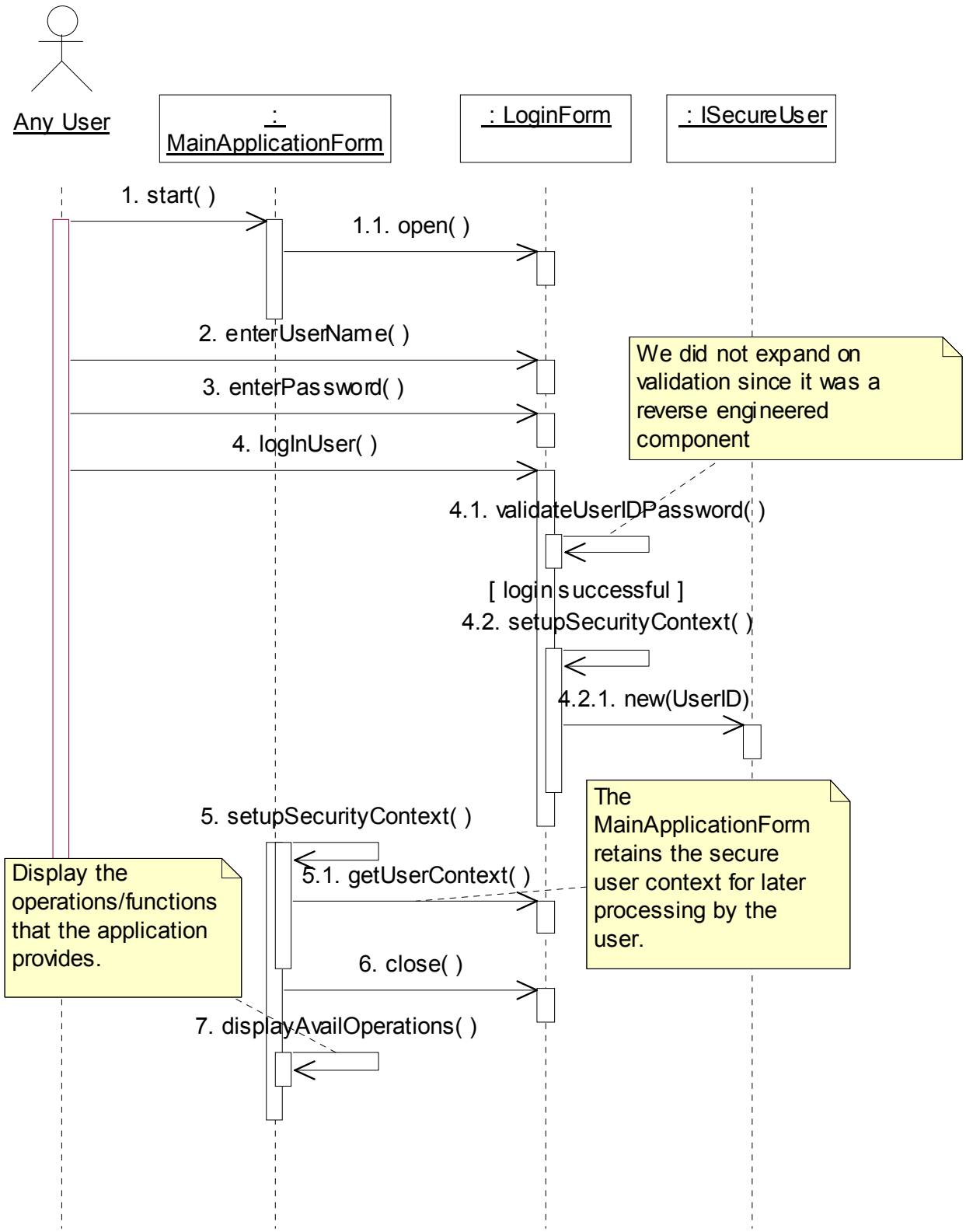
MainApplicationForm : Requirements Traceability:
- Usability: The desktop user-interface shall be Windows 95/98 compliant.

ISecureUser : Analysis Mechanisms:
- Security

LoginForm : Analysis Mechanisms:
- Security

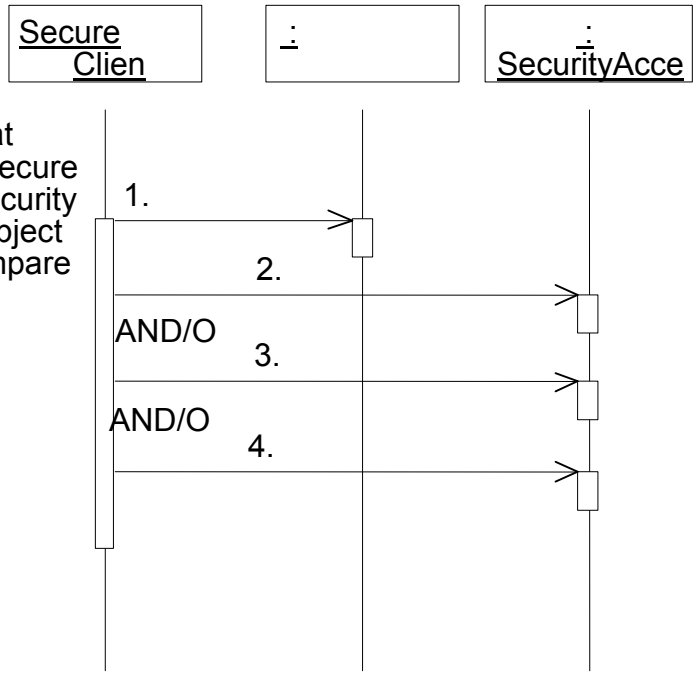
Requirements Traceability:
- Usability: The desktop user-interface shall be Windows 95/98 compliant.

Dynamic View: Secure User Set-Up



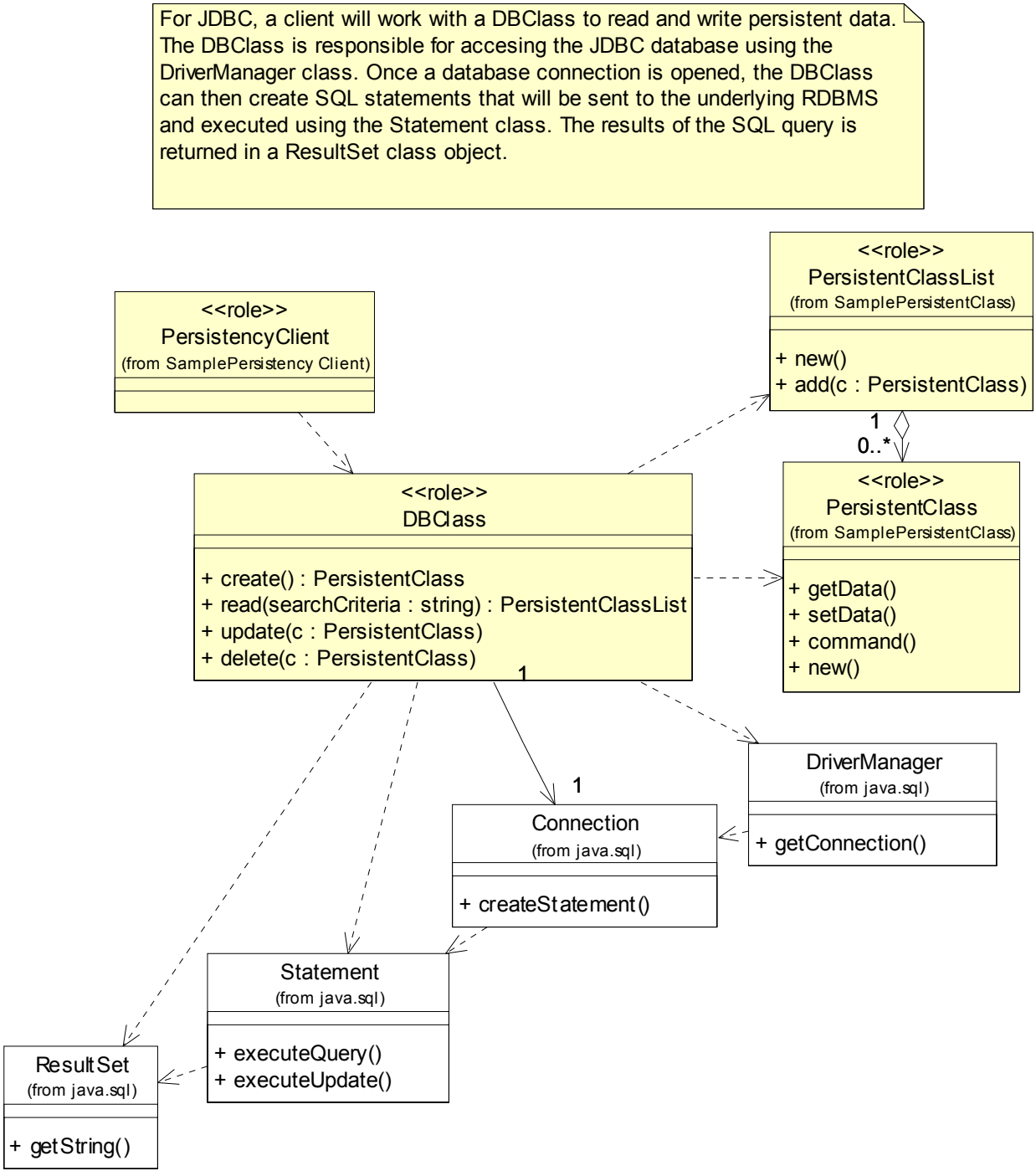
Dynamic View: Secure Data Access

After retrieving an object that SecureData interface, the Secure Client should retrieve the security information for the secure object for the current user and compare make sure the current user view/edit/delete the



Persistency - RDBMS - JDBC

Static View: Persistency JDBC



Class Descriptions

PersistencyClient : An example of a client of a persistent class.

PersistentClass : An example of a class that's persistent.

PersistentClassList :

Statement : The class used for executing a static SQL statement and obtaining the results produced by it. SQL statements without parameters are normally executed using Statement objects.

DBClass : A sample of a class that would be responsible for making another class persistent.
Every Class that's persistent will have a corresponding DBClass (e.g., Student will have a DBStudent class).

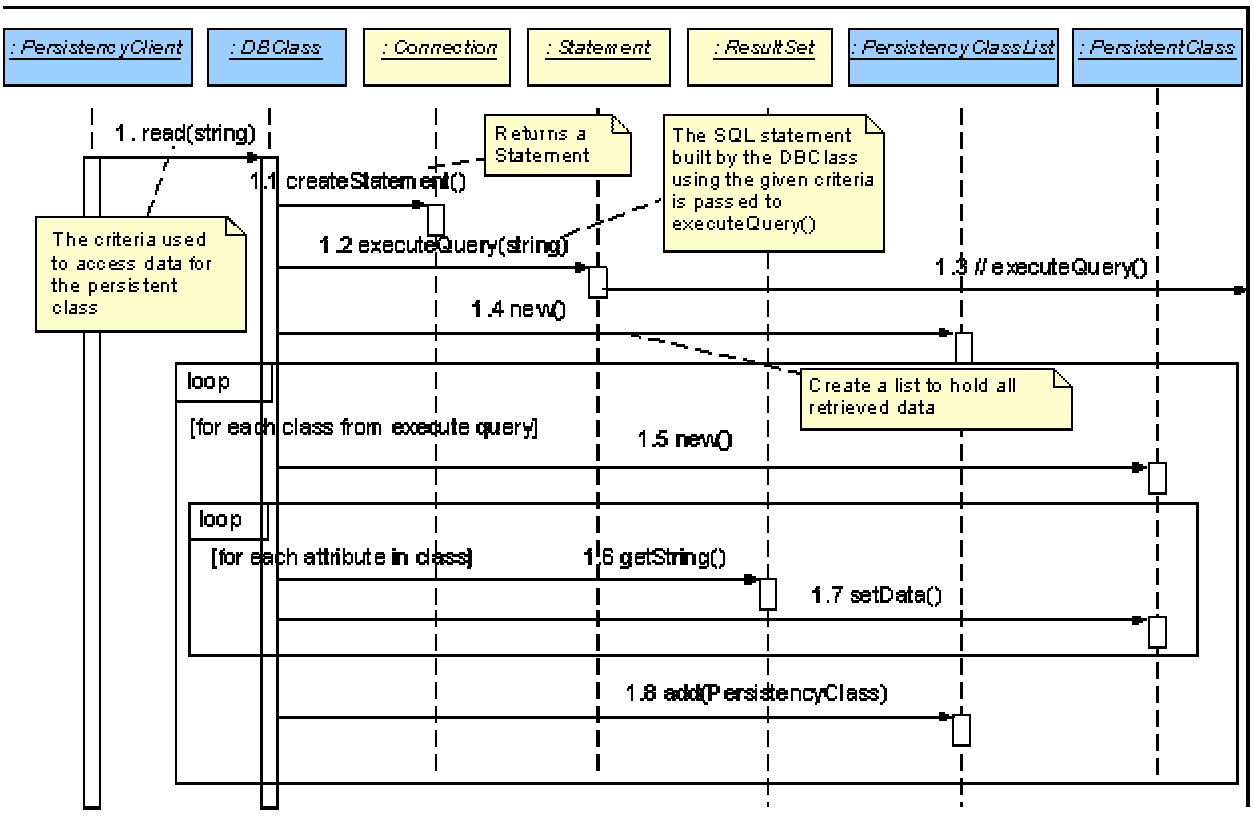
With an RDBMS, you need a mapping of objects/classes to tables, and you must recreate the (association/aggregation) structures. DBClass is a database interface class which understands the OO-to-RDBMS mapping and has the behavior to interface with the RDBMS. This database interface class is used whenever a persistent class needs to be created, accessed, or deleted. The database interface class flattens the object and writes it to the RDBMS and reads the object data from the RDBMS and builds the object.

Connection : A connection (session) with a specific database. Within the context of a Connection, SQL statements are executed, and results are returned.

ResultSet : A ResultSet provides access to a table of data. A ResultSet object is usually generated by executing a Statement.

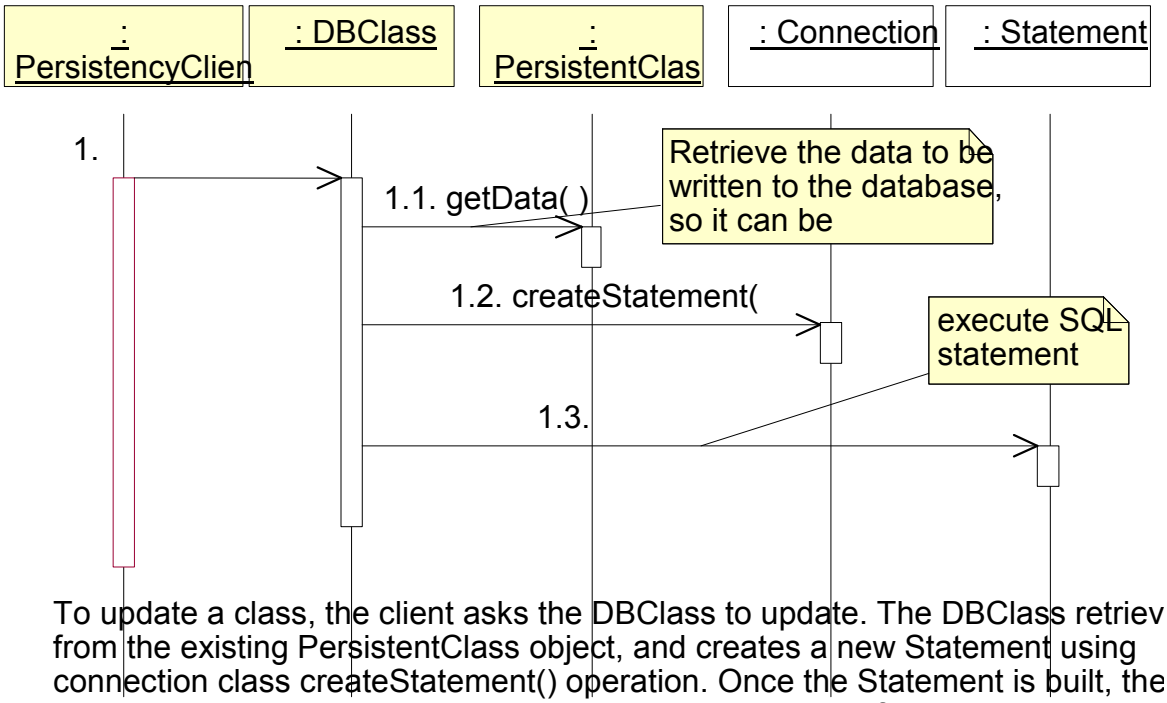
DriverManager : The basic service for managing a set of JDBC drivers.

Dynamic View: JDBC RDBMS Read



Dynamic View: JDBC RDBMS Update

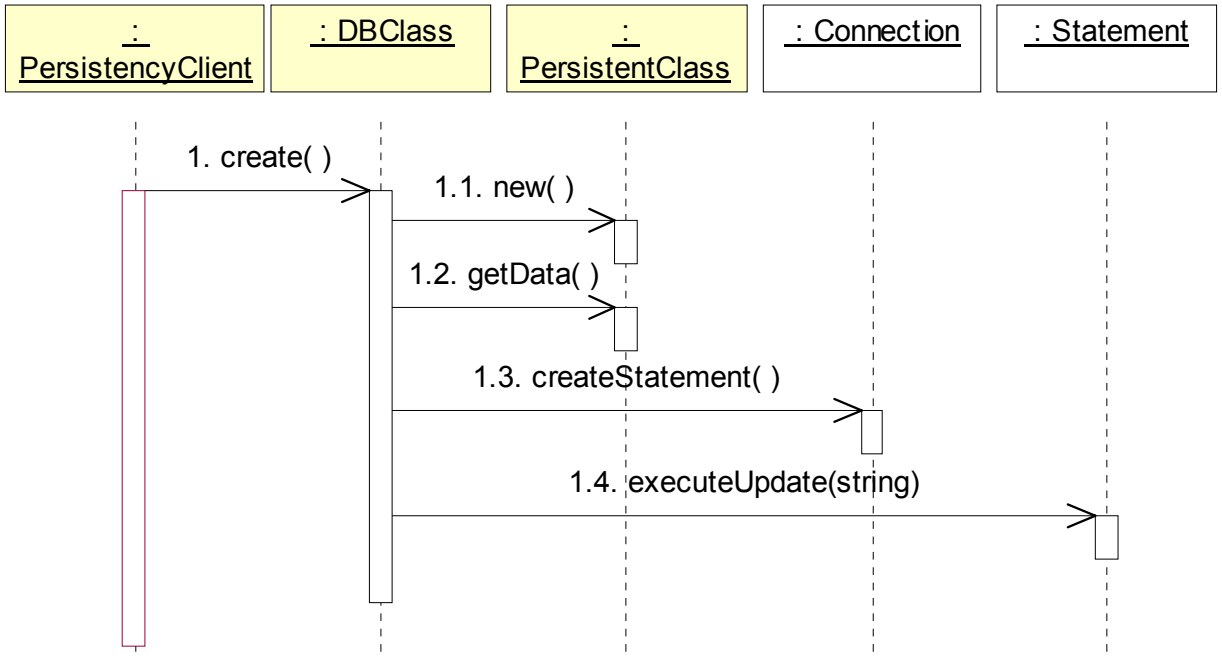
JDBC RDBMS



To update a class, the client asks the DBClass to update. The DBClass retriev from the existing PersistentClass object, and creates a new Statement using connection class createStatement() operation. Once the Statement is built, the executed, and the database is updated with the new data from the

Dynamic View: JDBC RDBMS Create

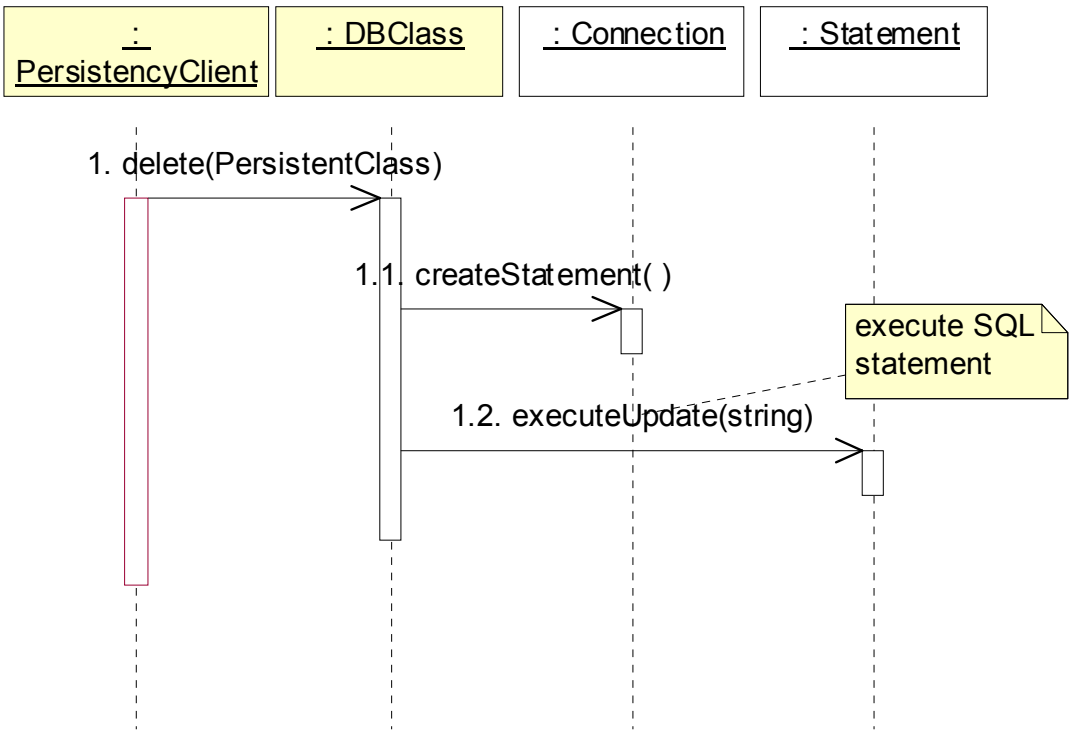
JDBC RDBMS Create



To create a new class, the client asks the DBClass to create the new class. The DBClass creates a new instance of Persistent Class with default values. The DBClass then creates a new Statement using the Connection class createStatement() operation. The statement is executed and the data is inserted into the database.

Dynamic View: JDBC RDBMS Delete

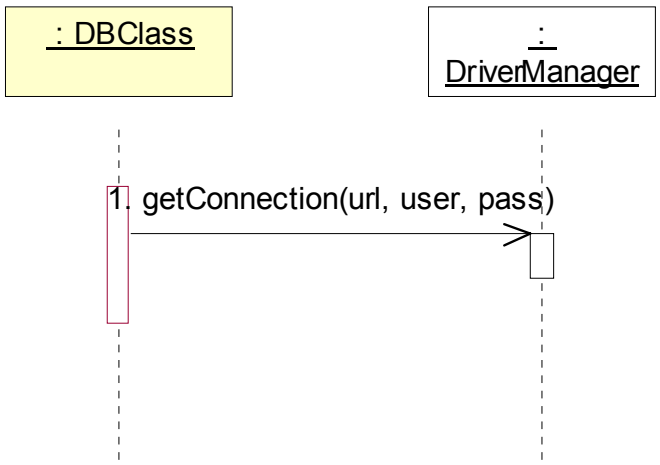
JDBC RDBMS Delete



To delete a class, the client asks the DBClass to delete a specific class instance. The DBClass creates a new statement using the Connection class createStatement() operation and formulates the correct SQL statement for the object instance that's passed in. The statement is executed and the data is removed from the database.

Dynamic View: JDBC RDBMS Initialize

JDBC RDBMS Initialize



To initialize the connection, the DBClass must load the appropriate driver by calling the DriverManager getConnection() operation with a URL, user, and password.

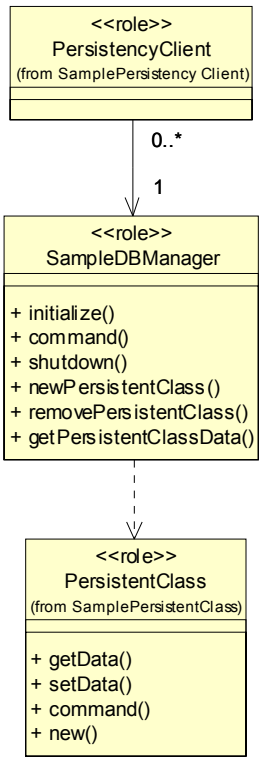
getConnection() attempts to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:
url - A database url of the form jdbc:subprotocol:subname
user - The database user on whose behalf the Connection is being made
password - The user's password

Returns a Connection to the URL

Persistency - OODBMS - ObjectStore

Static View: Persistency – ObjectStore OODBMS



Clients interface with the SampleDBManager class, which controls access to PersistentClass objects in the database. The SampleDBManager also controls user access, registration, and session management. The SampleDBManager might run as an application server that operates behind a web server and provides access to the database.

To access a persistent object, the client works with the SampleDBManager class. The client can create a new instance of the PersistentClass with the "newPersistentClass( )" operation, or invoke a command on the PersistentClass with a "command( )" operation. In a real application, the "command( )" operation would be replaced with operations from the PersistentClass.

The client is responsible for initializing and shutting down the database through the SampleDBManager class, however the client does not need to be aware of any of the details of the ObjectStore database.

In the context of the ObjectStore database, the PersistentClass is considered the "root class". If there were other root classes, there would be additional classes with association relationships with the SampleDBManager.

From the ObjectStore manual: "Objects become persistent when they are referenced by other persistent objects. The application defines persistent roots and when it commits a transaction, PSE/PSE Pro finds all objects reachable from persistent roots and stores them in the database.
This is called persistence by reachability and it helps to preserve the automatic storage management semantics of Java."

You define the PersistentClass for persistent use the same way you define it for transient use. Other than the required import com.odi.* statement, there is almost no special code for persistent use of the PersistentClass.

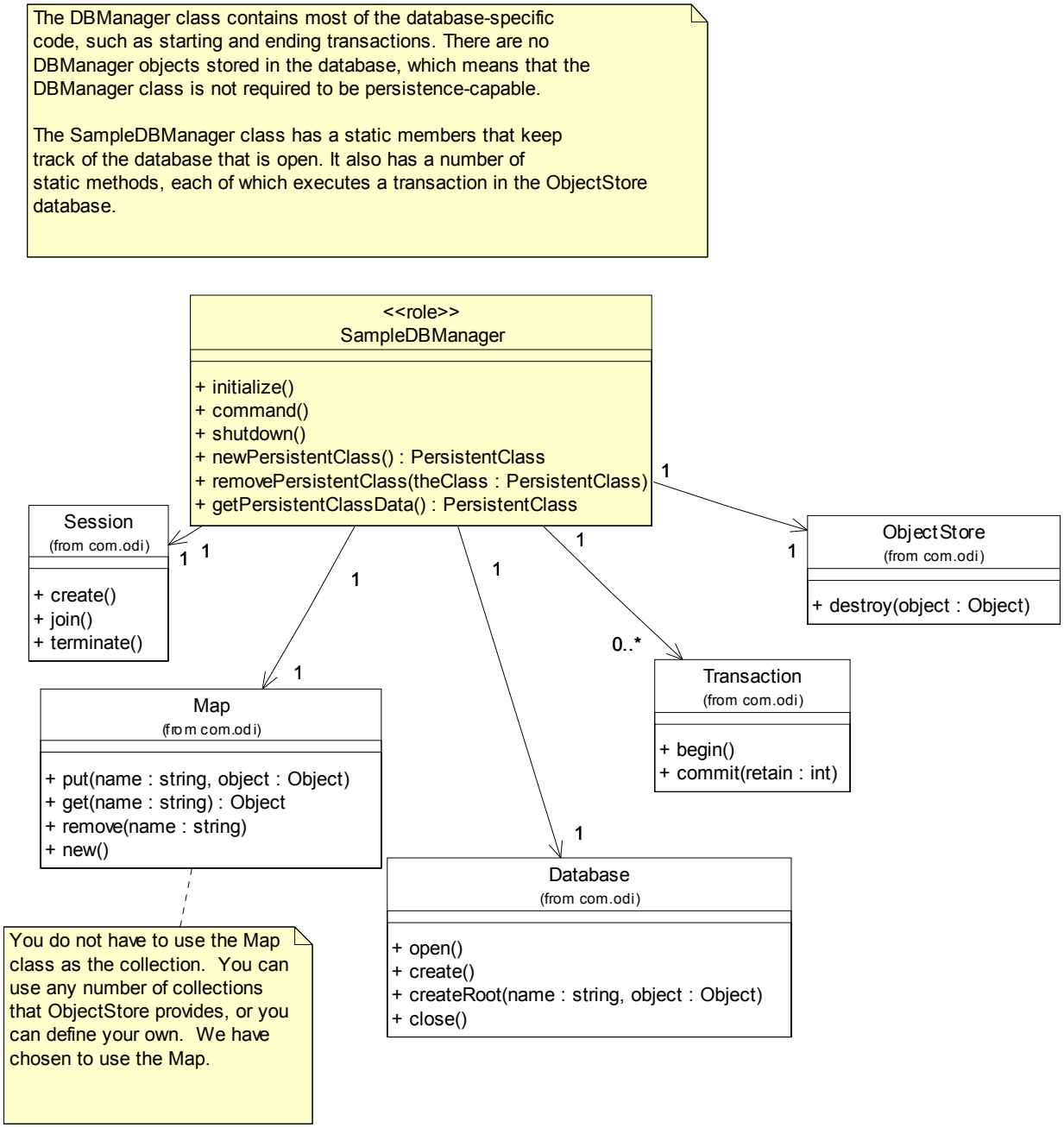
Class Descriptions

PersistencyClient : An example of a client of a persistent class.

SampleDBManager : Responsible for providing access to the persistent objects.
The SampleDBManager is an example of a class an ObjectStore user would write. It is a control class that provides a single entry point into a specific ObjectStore database. The user would add the appropriate operations to the class to access entities in the database. It is often implemented as a singleton, but doesn't have to be (if an application needs to have multiple instances of a database open at once, then each instance would have its own SampleDBManager). Both solutions would work, it just depends on how you want to do it.

PersistentClass : An example of a class that's persistent.

Static View: Persistency - DBManager Detail



Class Descriptions

Session : The class that represents a database session. A session must be created in order to access the database and any persistent data.

A session is the context in which PSE/PSE Pro databases are created or opened, and transactions can be executed. Only one transaction at a time can exist in a session.

Map : A persistent map container classes that stores key/value pairs.

Database : The Database class represents an ObjectStore database.

Before you begin creating persistent objects, you must create a database to hold the objects. In subsequent processes, you open the database to allow the process to read or modify the objects. To create a database, you call the static create() method on the Database class and specify the database name and an access mode.

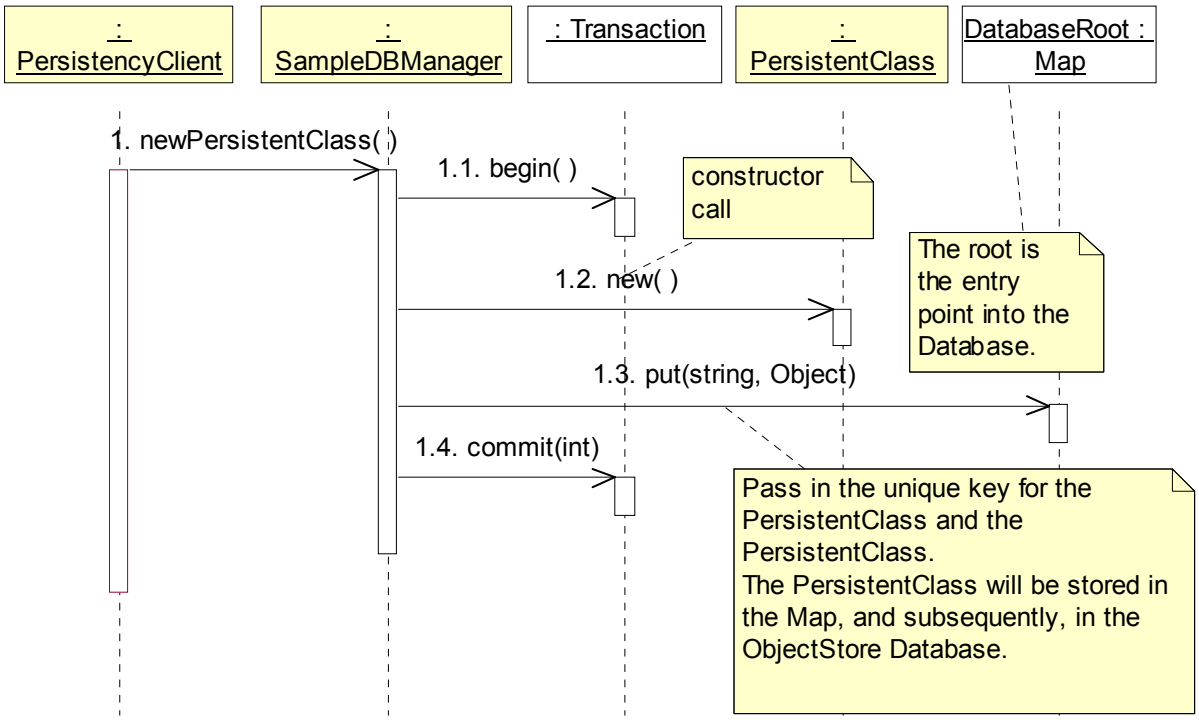
Transaction : An ObjectStore transaction. Manages a logical unit of work. All persistent objects must be accessed within a transaction.

ObjectStore : Defines system-level operations that are not specific to any database.

SampleDBManager : Responsible for providing access to the persistent objects.
The SampleDBManager is an example of a class an ObjectStore user would write. It is a control class that provides a single entry point into a specific ObjectStore database. The user would add the appropriate operations to the class to access entities in the database. It is often implemented as a singleton, but doesn't have to be (if an application needs to have multiple instances of a database open at once, then each instance would have it's own SampleDBManager). Both solutions would work, it just depends on how you want to do it.

Dynamic View: ObjectStore – OODBMS Create

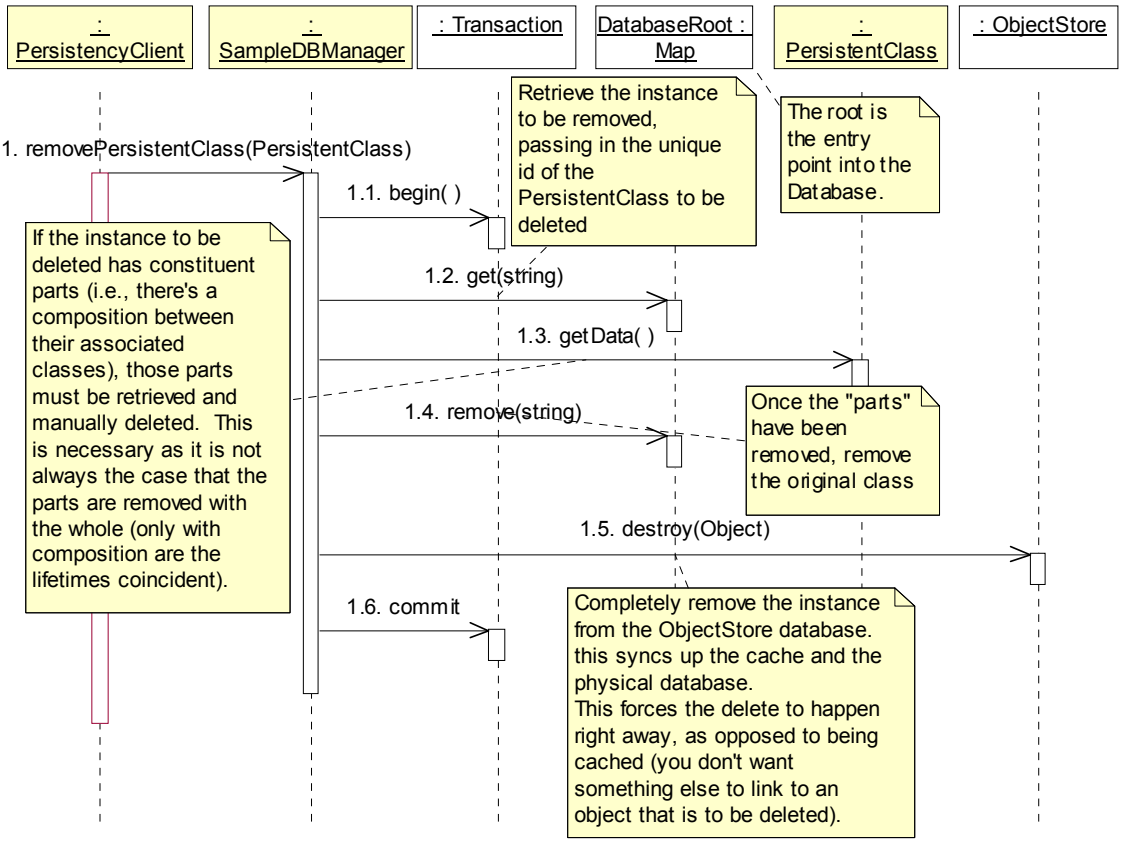
ObjectStore OODBMS Create



To create a new instance of PersistentClass in the database, the SampleDBManager first creates a transaction and then calls the constructor for PersistentClass. Once the class has been constructed the class is added to the database via the root "put()" operation. The transaction is then committed.

Dynamic View: ObjectStore OODBMS Delete

ObjectStore OODBMS Delete

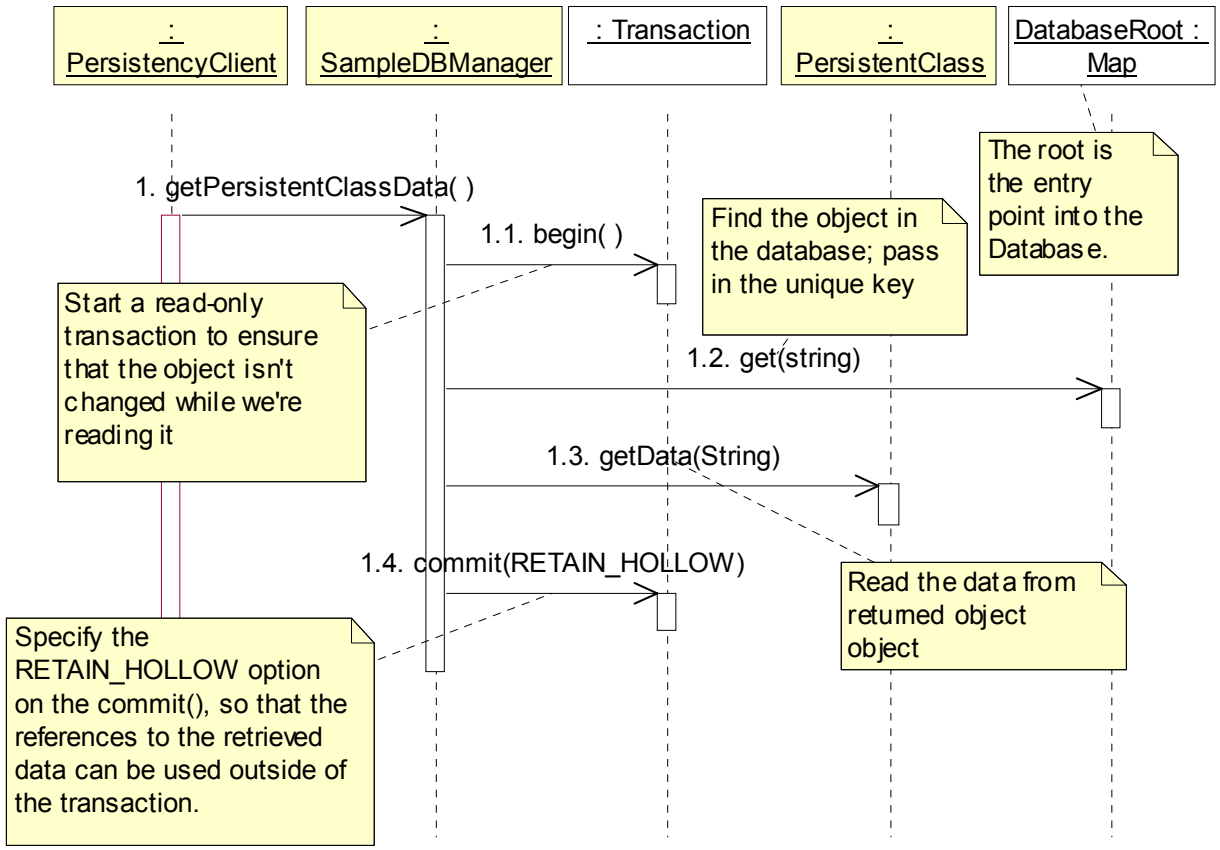


To delete an object from the database, the SampleDBManager first creates a new transaction, removes any constituent parts, and then removes the object using the database root "remove()" operation. The object is then completely removed from the ObjectStore database immediately via ObjectStore.destry (). Once the object has been removed, the transaction is committed.

Thus, in ObjectStore, delete really has two steps -- removal from the container class that is the database in memory, and removal from the physical database. that is because you want the deletion to occur right away, as opposed to being cached.

Dynamic View: ObjectStore OODBMS Read

ObjectStore OODBMS Read

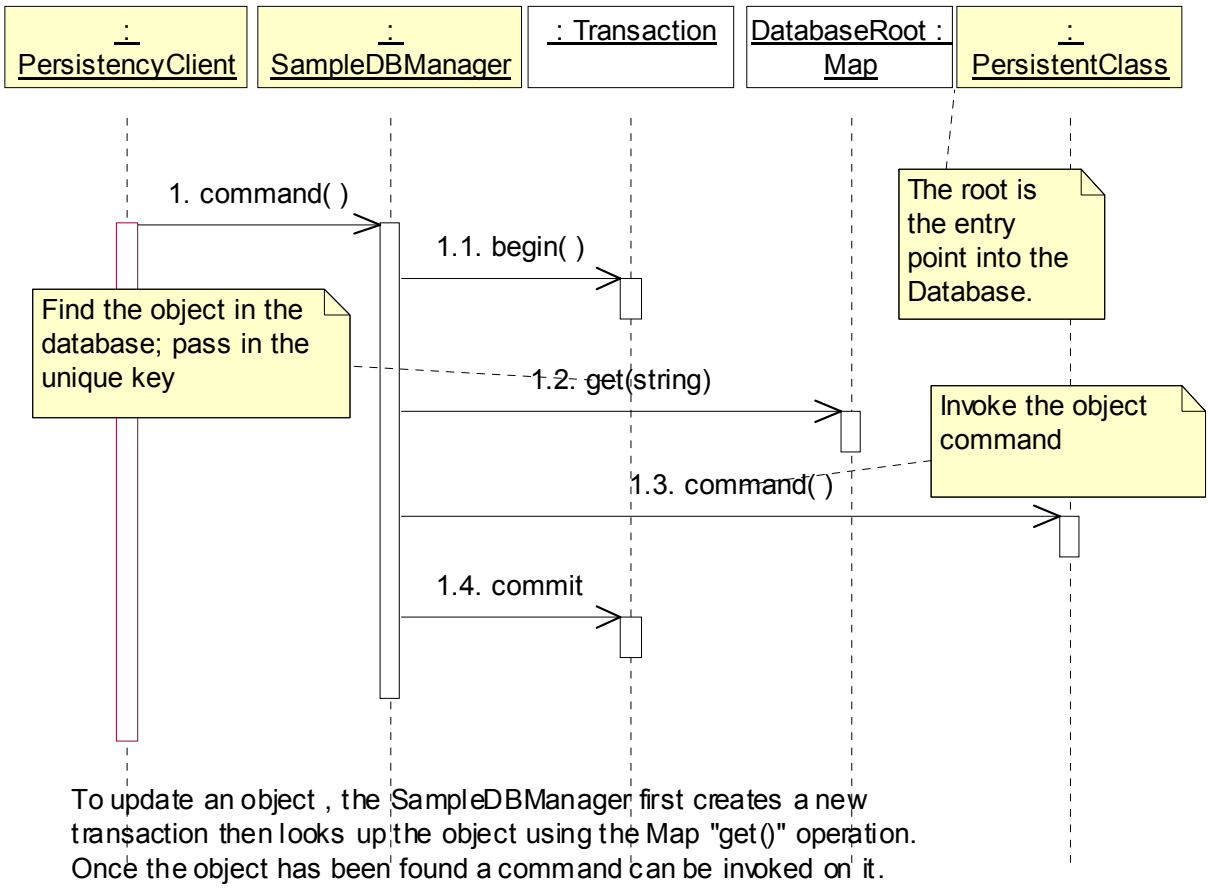


To read an object , the SampleDBManager first creates a new read-only transaction then looks up the object using the Map "get()" operation. Once the object has been found it can be read with the "getData()" operation, and the transaction committed. RETAIN_HOLLOW is specified for the commit,. so the references to the object and the retrieved data can be used outside of the retrieval transaction. Once the transaction is committed the object can then be updated.

Note: Even though RETAIN_HOLLOW is specified, it does not guarantee the integrity of the reference outside of the transaction. There is still some risk that the reference could be outdated. RETAIN_HOLLOW basically says "I'm consciously taking such a risk". If that option was not used, then the references would not be available.

Dynamic View: ObjectStore OODBMS Update

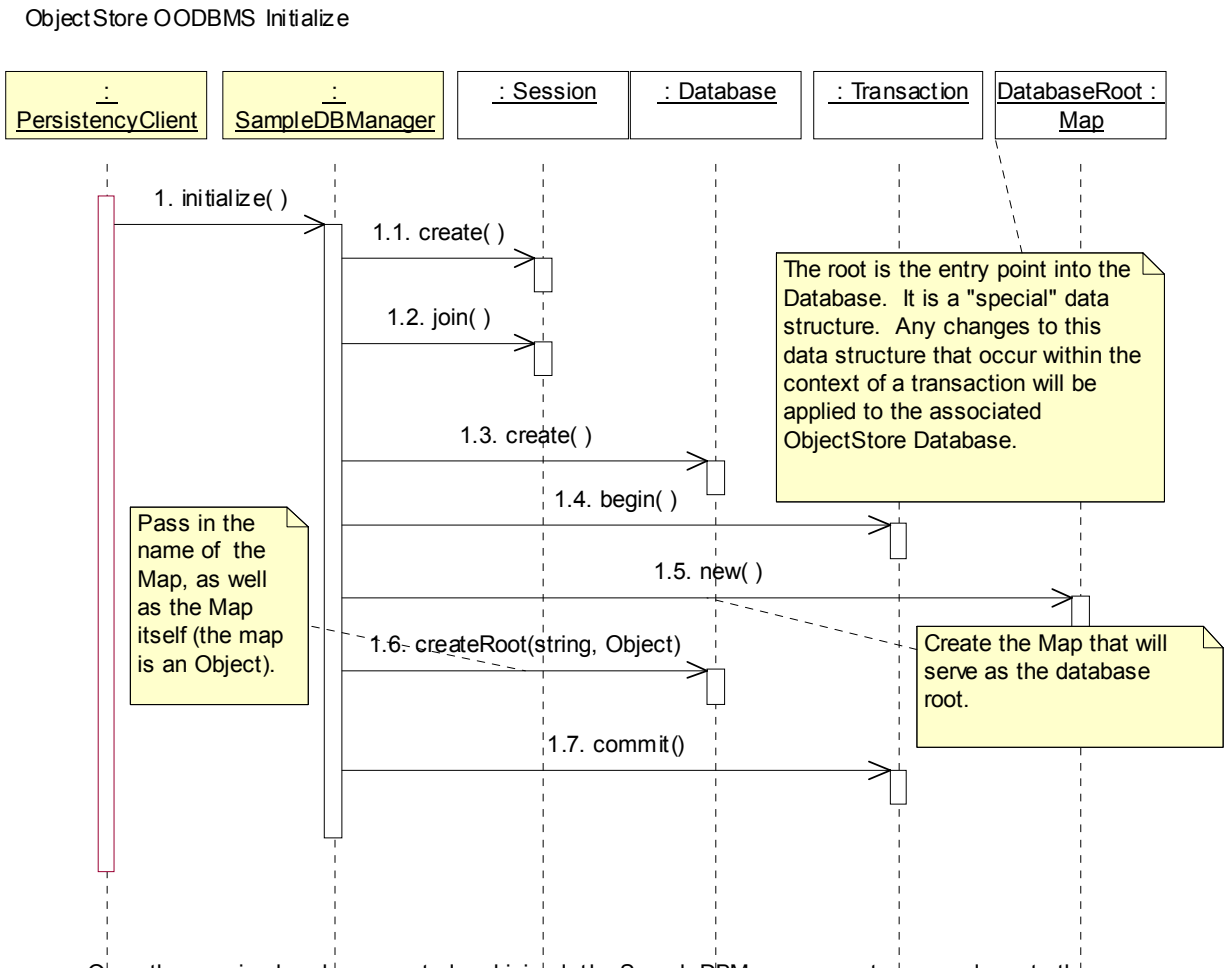
ObjectStore OODBMS Update



To update an object , the SampleDBManager first creates a new transaction then looks up the object using the Map "get()" operation. Once the object has been found a command can be invoked on it. When the command is complete the transaction is committed.

A separate put() to the Map is not necessary as the get() operation returns a reference to the persistent object and any changes to that object, if made in the context of a transaction, are automatically committed to the database.

Dynamic View: ObjectStore OODBMS Initialize



Once the session has been created and joined, the SampleDBManager must open and create the new database.

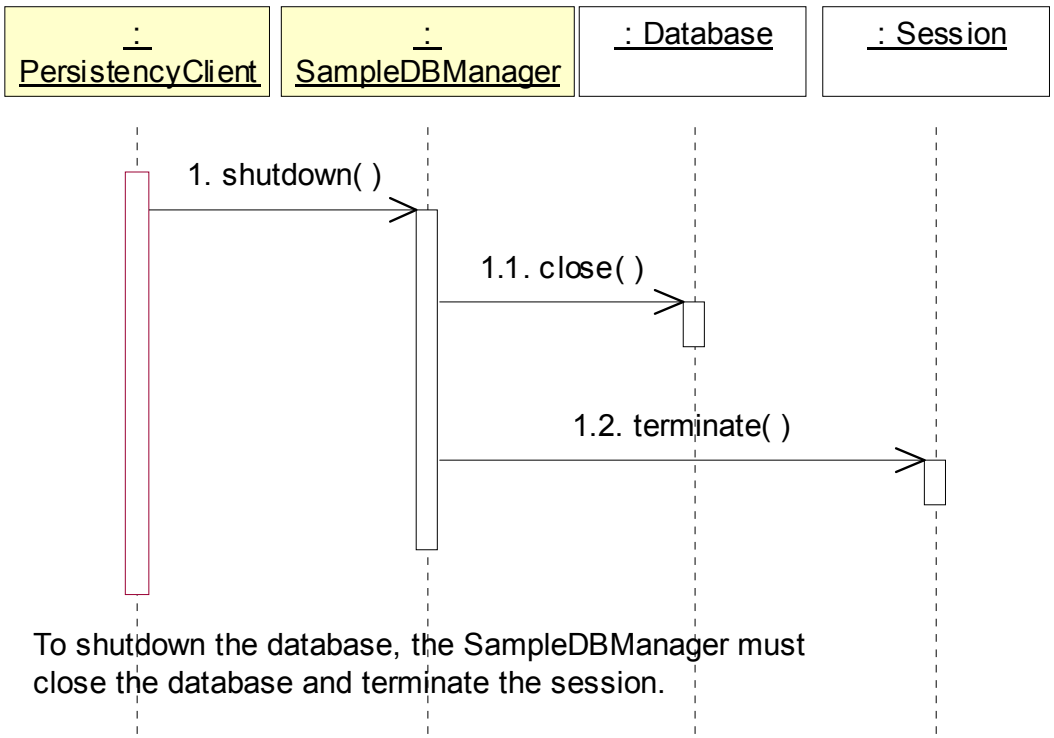
To create the database, the SampleDBManager creates a new transaction and creates the "root" of the database with the "createRoot()" operation.

The root is the entry point into the Database (the root class is the top-level class in the object database). It is a "special" data structure (in the above example, a Map that contains instances of the root class and all “reachable” classes). Any changes to this data structure that occur within the context of a transaction will be applied to the associated ObjectStore Database. There may be multiple database roots.

Once the root has been created, the transaction is committed.

Dynamic View: ObjectStore OODBMS Shutdown

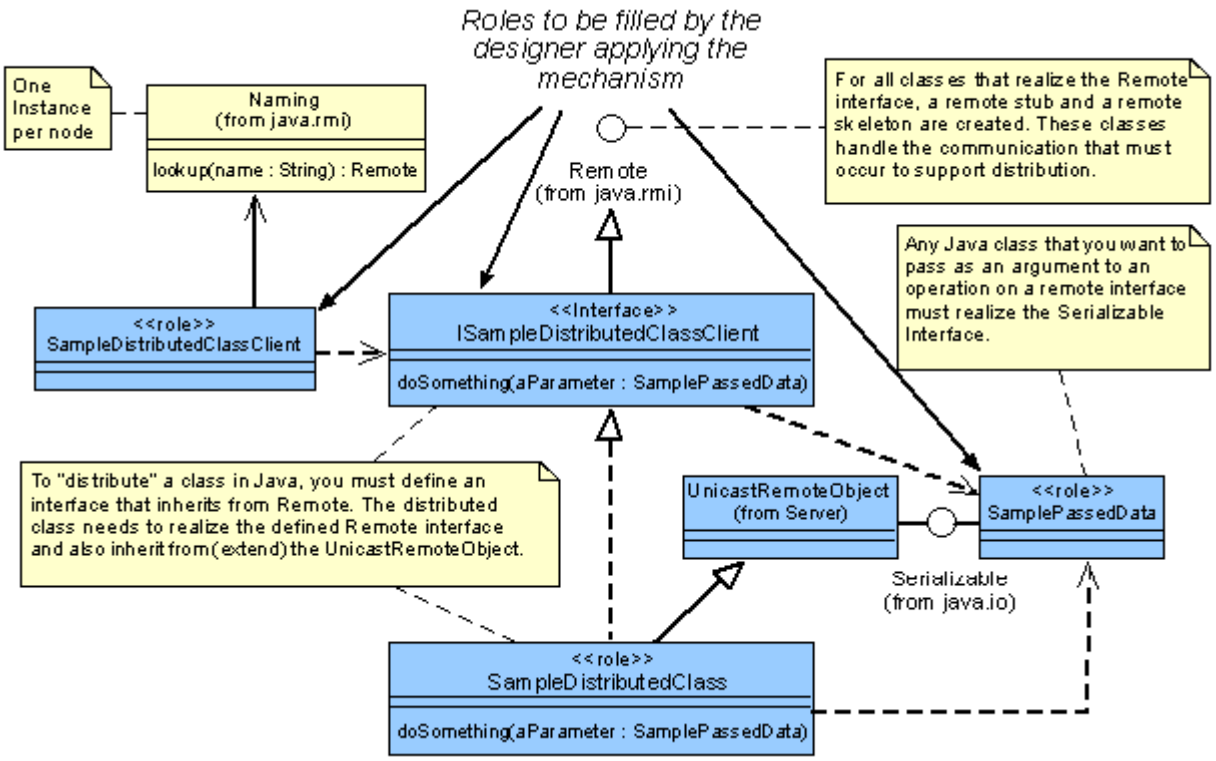
ObjectStore OODBMS Shutdown



To shutdown the database, the SampleDBManager must close the database and terminate the session.

Distribution - RMI

Static View: Distribution - RMI



Class Descriptions

Naming. :
* This is the bootstrap mechanism for obtaining references to remote
* objects based on Uniform Resource Locator (URL) syntax. The URL
* for a remote object is specified using the usual host, port and
* name:
*
 rmi://host:port/name
*
 host = host name of registry (defaults to current host)
*
 port = port number of registry (defaults to the registry port number)
*
 name = name for remote object

SampleDistributedClass : An example of a class that's distributed.

Remote :
* The Remote interface serves to identify all remote objects.
* Any object that is a remote object must directly or indirectly implement
* this interface. Only those methods specified in a remote interface are
* available remotely. <p>
* Implementation classes can implement any number of remote interfaces
* and can extend other remote implementation classes.

For all classes that realize the Remote interface, a remote stub and a remote skeleton are created. These classes handle the communication that must occur to support distribution.

SampleDistributedClassClient : An example of a client of a distributed class.

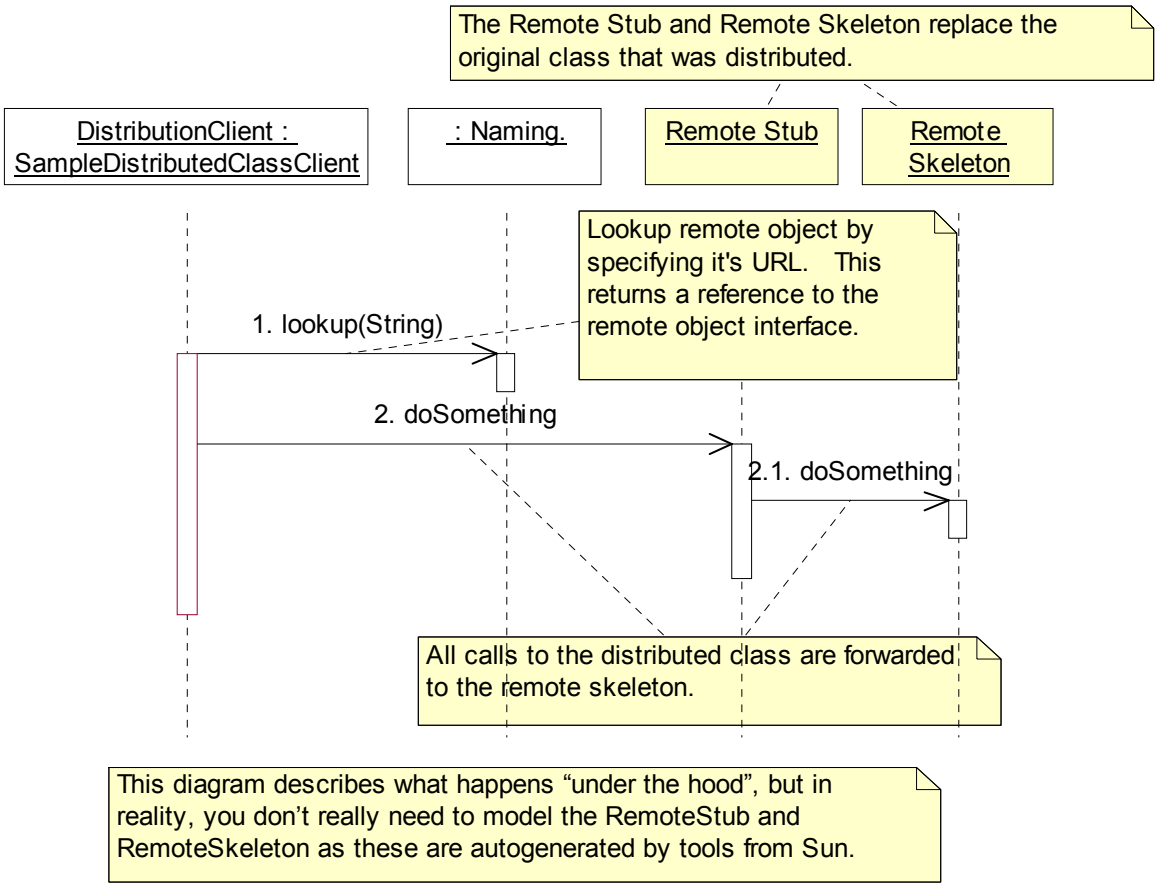
SamplePassedData : An example of data that is passed to/from a distributed class.

UnicastRemoteObject :

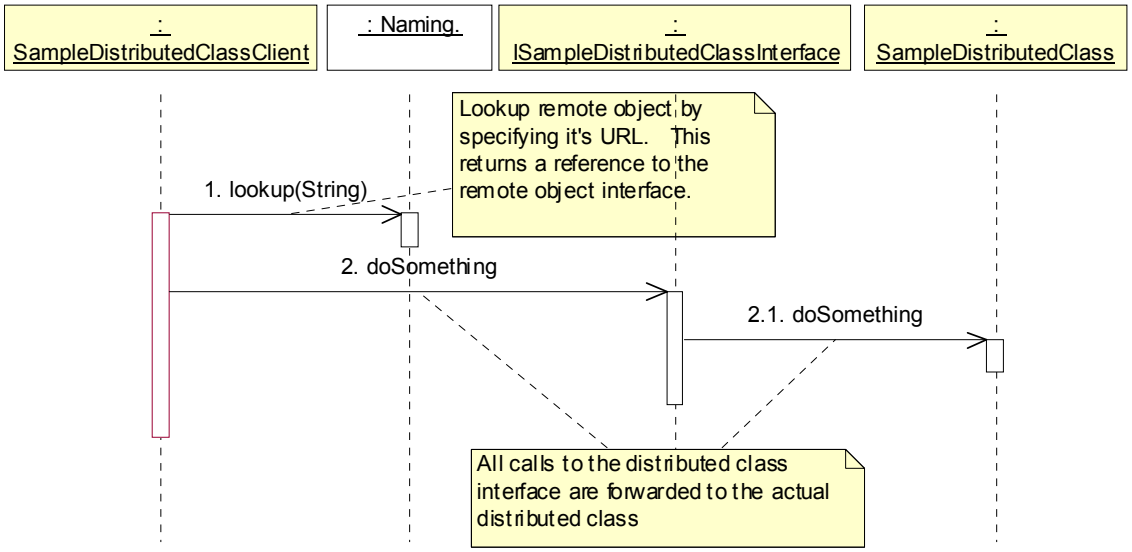
ISampleDistributedClassInterface : An example of an interface defined for a distributed class.

Serializable : Any Java class that you want to pass as an argument to an operation on a remote interface must realize the Serializable interface.

Dynamic View: Set Up Remote Connection (details)



Dynamic View: Set Up Remote Connection



Logical View

Architectural Analysis

Upper-Level Layers

- Application layer
- Business Services layer

Upper-Level Layer Dependencies

- The Application layer depends on the Business Services layer

Architectural Design

Incorporating ObjectStore

For the Payroll System, a single root class has been chosen -- Employee.

The selected container is the Map, where the unique key to access the Employees is EmployeeID.

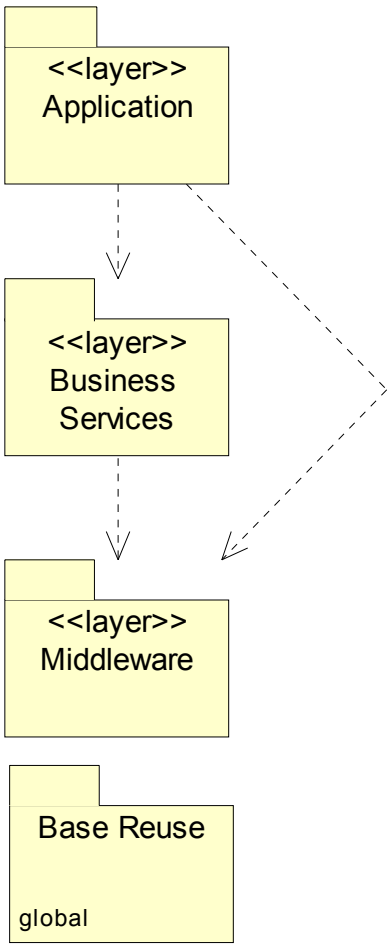
There is one DBManager class per ObjectStore database instance. For the Payroll System, there is one ObjectStore database, the Payroll Database, that contains employee information, including timecard, purchase order, and paycheck information. Thus, there is one PayrollDBManager that exists in the new OODBMS Support package.

For the ObjectStore persistency mechanism, the DBManager class includes operations to access the OODBMS persistent entities in the database. For the PayrollDBManager class, operations have been added to access Employee, Timecard, Purchase Order, and Paycheck information since that is required for the core system functionality.

During Identify Design Mechanisms, the architect provides guidance to the designers and makes sure that the architecture has the necessary infrastructure to support the mechanism. Thus, the PayrollDBManager and the supporting architectural packages and relationships (OODBMS Support) have been defined in Identify Design Mechanisms. However, the development of the interaction diagrams that describe these operations and where they fit into the existing use-case realizations has been deferred until detailed design (e.g., Use-Case and Subsystem Design).

The following diagram demonstrates the operations that have been defined for the PayrollDBManager during Identify Design Mechanisms:

Architectural Layers and Their Dependencies: Main Diagram



Layer Descriptions

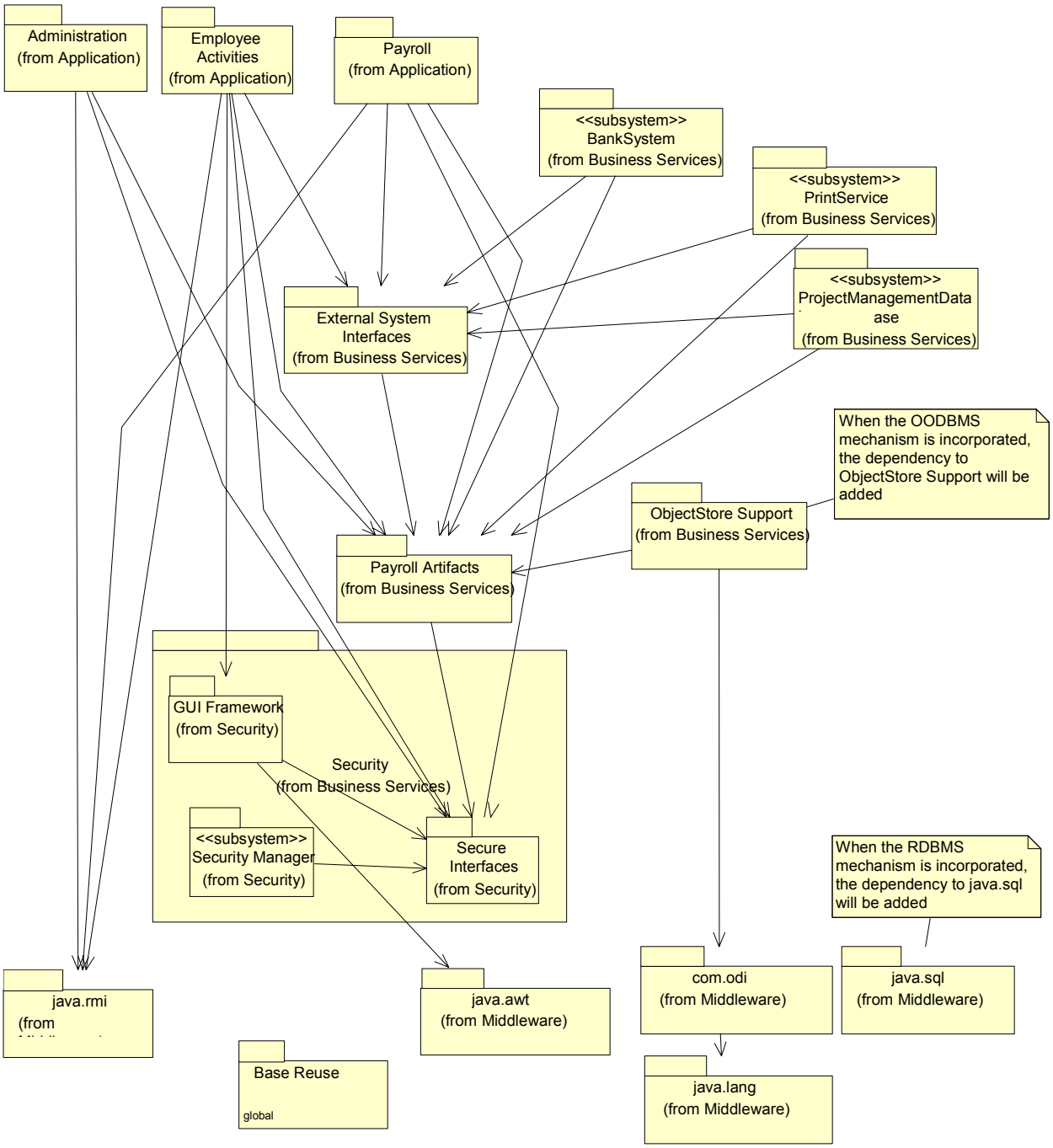
Application Layer: The Application layer contains application-specific design elements.

Business Services Layer: The Business Services layer contains business-specific elements that are used in several applications.

Base Reuse : Basic reusable design elements.

Middleware Layer: Provides utilities and platform-independent services.

Packages and Their Dependencies: Package Dependencies Diagram



Package Descriptions

Employee Activities : Contains the design elements that support the Employee's applications.

Administration : Contains the design elements that support the Payroll Administrator's applications.

Payroll : Contains the design elements that support the execution of the payroll processing.

Payroll Artifacts : Contains the core payroll abstractions.

BankSystem Subsystem: Encapsulates communication with all external bank systems.

External System Interfaces : Contains the interfaces that support access to external systems. This is so that the external system interface classes can be version controlled independently from the subsystems that realize them.

PrintService Subsystem: Provides utilities to produce hard-copy.

ProjectManagementDatabase Subsystem: Encapsulates the interface to the legacy database containing information regarding projects and charge numbers.

java.awt : The java.awt package contains the basic GUI design elements for java.

com.odi : The com.odi package contains the design elements that support the OODBMS persistency mechanism. The name of the package in the model reflects the naming convention for 3rd party Java software. The convention is to use the reverse of the domain name, so if Rational had a Java package called "util" they'd call it "com.rational.util". This com.odi has nothing to do with Microsoft COM/DCOM; they are totally separate. There is nothing COM/DCOM related when using CORBA, RMI, or ObjectStore.

Base Reuse : Basic reusable design elements.

java.lang : The package contains some basic java design elements.

Security : Contains design elements that implement the security mechanism.

GUI Framework : This package comprises a whole framework for user interface management.

It has a ViewHandler that manages the opening and closing of windows, plus window-to-window communication so that windows do not need to depend directly upon each other.

This framework is security-aware, it has a login window that will create a server-resident user context object. The ViewHandler class manages a handle to the user context object.

The ViewHandler also starts up the controller classes for each use case manager.

Secure Interfaces : Contains the interfaces that provide clients access to security services.

Security Manager Subsystem: Provides the implementation for the core security services.

ObjectStore Support : Contains the business-specific design elements that support the OODBMS persistency mechanism. This includes the DBManager. The DBManager class must contain operations for every OODBMS persistent class.

java.rmi : The java.rmi package contains the classes that implement the RMI distribution mechanism. This package is commercially available with most standard JAVA IDEs.

java.sql : The package that contains the design elements that support RDBMS persistency.

Process View

Processes

The processes for the Payroll System will be the following:

One process per major interface or family of forms (e.g. EmployeeApplication):
- EmployeeApplication: Controls the interface of the Employee application. Controls the family of forms that the employee uses.

There is one process per major interface because these are now seen as separate, mutually exclusive applications that will run concurrently with each other.

One process per business service controller:
- PayrollControllerProcess
- TimecardControllerProcess

There is one process per controller because these activities will need to run concurrently with each other.

One process per external system:
- ProjectManagementDBAccess
- BankSystemAccess
- PrinterAccess

There is one process per external system. These processes manage access to those systems. Such access may be slow, so this allows other functionality to continue while the external system processes wait on the external system. These processes also synchronize access to the external systems from the other system processes.

To further improve throughput and turnaround, a Bank Transaction thread was defined to allow multiple accesses to the Bank System to occur concurrently. Each time a transaction needs to be sent to the Bank System, a different thread is used. The Bank Transaction thread will run in the context of the Bank System Access process.

In general, the above processes and threads were defined to support faster response times and take advantage of multiple processors.

Design Element to Process Mapping

- The classes associated with the individual user interfaces should be mapped to those processes.
- The classes associated with the individual business services should be mapped to those processes.
- The classes associated with access to the external systems should be mapped to those processes.

Deployment View

Nodes and Connections

The nodes of the physical architecture for the Payroll System are the following:
- Desktop PCs (processors)
- Payroll Server (processor)
- Bank System (processor)
- Project Management Database (processor)
- Printers (devices)

- The Desktop PCs are connected to the Payroll Server via the Company LAN
- The Printers are connected to the Payroll Server via the Company LAN
- The Payroll Server is connected to the external Bank System via the Internet.
- The Payroll Server is connected to the ProjectManagementDatabase via the Company LAN

Process-to-Node Map

The following processes run on the Desktop PCs:
- EmployeeApplication

The following processes run on the Payroll Server:
- PayrollControllerProcess
- TimecardControllerProcess
- ProjectManagementDBAccess
- BankSystemAccess
- PrinterAccess

