

Object-Oriented Analysis and Design with UML2 and Rational Software Modeler

Student Workbook

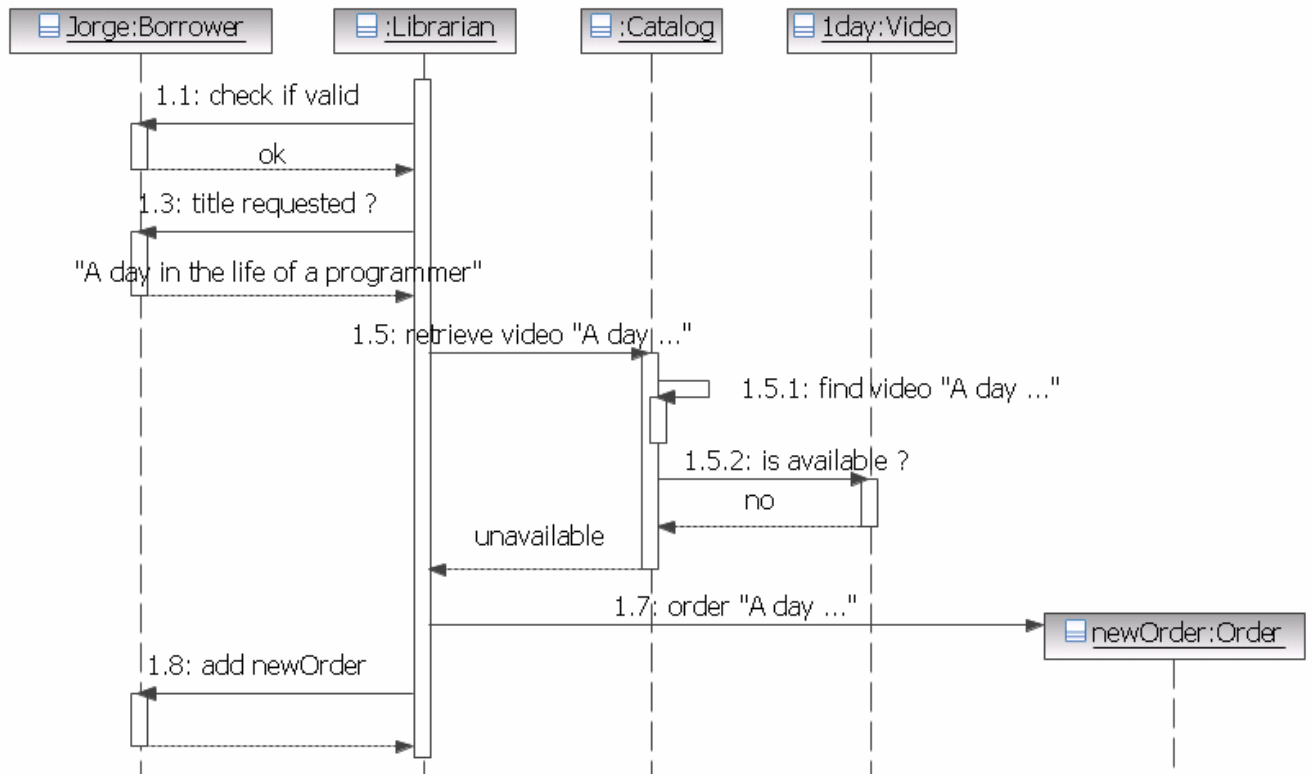
Table of contents

Lab 1 – Sequence Diagram	3
Lab 2 – Class Diagrams	4
Task 2.1: Modeling graphs, points and connectors	4
Task 2.2: Modeling a family tree	4
Task 2.3: Modeling a file system	4
Lab 3 – Requirements Management.....	5
Task 3.1: Preliminary setup.....	5
Task 3.2: Identify the actors and use cases of the payroll system.....	10
Lab 4 – Architectural Analysis.....	12
Task 4.1: Creating the Analysis Model	12
Task 4.2: Identify the Key Abstractions of the Payroll System	14
Task 4.3: Represent the Higher Layers of the Architecture	16
Lab 5 – Use Case Analysis.....	17
Task 5.1: Create Use-Case Realizations	17
Task 5.2: Assign the Use-Case Behavior to Classes	18
Task 5.3: Complete the VOPC	19
Open the VOPC diagram for the Maintain Timecard UC Realization.	19
Task 5.4: Map Analysis Mechanisms to Classes	19
Lab 6 – Identify Design Elements	20
Task 6.1: Transform an Analysis Class into a Subsystem	20
Lab 7 – Class Design.....	22
Lab 8 – Subsystem Design	23
Lab 9 – Finalize the Design Model	24
Appendix 1 – Course Registration Requirements	25
Problem Statement	25
Glossary.....	26
Supplementary Specifications	27
Register for Courses UC	29
Appendix 2 – Payroll System.....	32
Problem Statement	32
Glossary.....	34
Supplementary Specifications	35
Maintain Timecard UC.....	36
Run Payroll UC	38



Lab 1 – Sequence Diagram

Interpret the following sequence diagram:





Lab 2 – Class Diagrams

Task 2.1: Modeling graphs, points and connectors

Create a class diagram to model the following concepts:

1. A drawing is composed of several graphs.
2. A graph is composed of one or more points, that may be connected or not.
3. One connector connects two points. There can be only one connector between two points.
4. Every point has a color.
5. Every connector has a color.
6. All the connectors of a given graph have the same color.
7. All the points of a given graph have the same color.
8. Destroying a point also destroys the associated connectors.

You can complete these statements with your own assumptions.

Task 2.2: Modeling a family tree

Model in a class diagram a family tree: the fact that one has parents and possibly children. Add relevant attributes.

Focus on the “biological” family ties – one has always two parents, even if they are deceased.

Consider creating a first model without using generalizations, and a second one with generalizations.

Hint: It might be necessary to add constraints to accurately model the family tree.

Task 2.3: Modeling a file system

Model a simple file system: a directory may contain other directories and/or files.

Implement the *destroy()* operation for both directories and files. Assume that a file or directory can destroy itself by calling a hypothetical system function *System.destroy(self)*. However, in the case of a directory, this function can be called only if the directory is empty. For a file, the *destroy()* operation looks like (in Java-like pseudo-code):

```
public class File {
    public void destroy() {
        System.destroy(self);
    }
}
```



Lab 3 – Requirements Management

Task 3.1: Preliminary setup

For the remaining labs, the following files are provided:

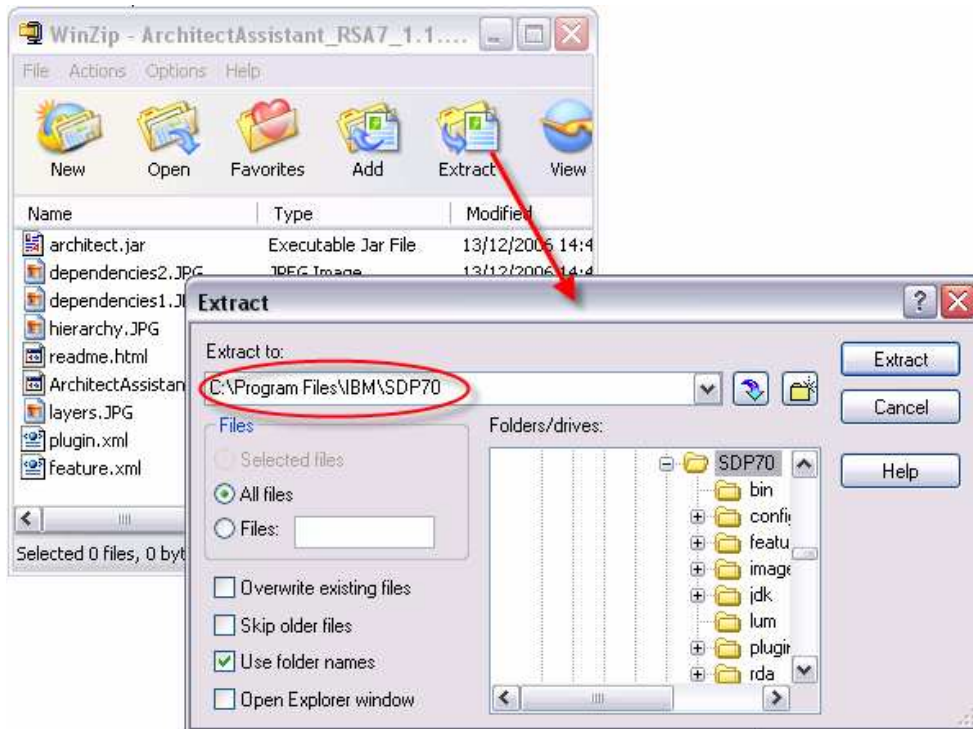
- *ArchitectAssistantxxxxxx.zip* contains a plug-in that controls dependencies between model components. This plug-in is used in the Design section of the course.
- *PayrollSystem.zip* contains the solutions to the different labs (starting from lab 3). Please do NOT open the RSM/RSA models before being told to.

Installing the ArchitectAssistant plug-in:

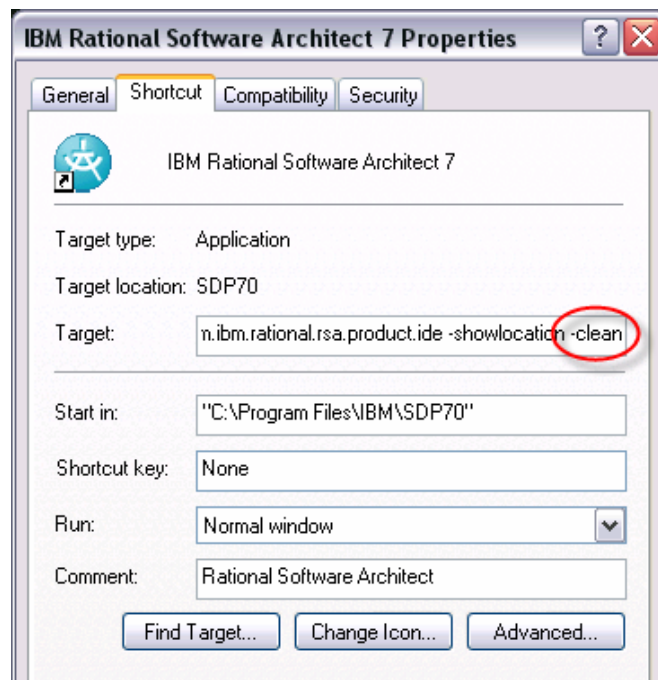
1. Extract the contents of *ArchitectAssistantxxxxxx.zip* in a valid RSM extension location:
 - a. An extension location is where plug-ins are installed. You can see all extension locations by using the command *Help > Software Updates > Manage Configuration* (this may take a long time). The snapshot below shows the default configuration settings: you should use *C:\Program Files\IBM\SDP70* as the *C:\Program Files\IBM\SDP70Shared* location is not updatable.



- b. In the example below, the contents of the zip file will be extracted into the *C:\Program Files\IBM\SDP70* directory:

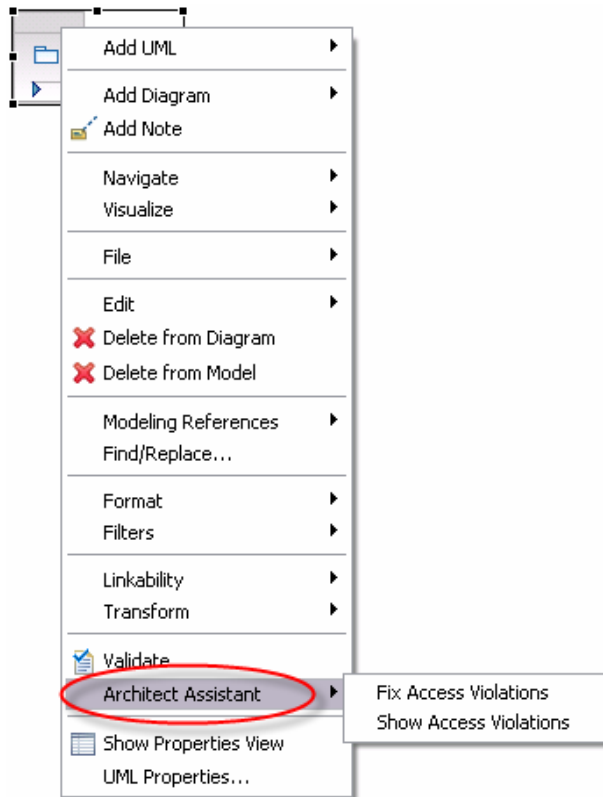


2. To ensure that RSA/RSM picks up the newly added plug-in, you must restart RSA/RSM with the *-clean* option
 - a. To add the *-clean* option, create (or edit) the RSM/RSA shortcut and add “-clean” as shown below:



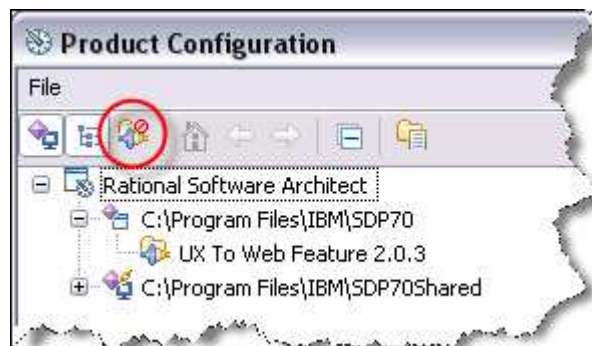
- b. Restart RSM/RSA.


- c. To verify that the plug-in was correctly added, do a right-click on any UML package or model. You should have an *ArchitectAssistant* submenu as shown below:



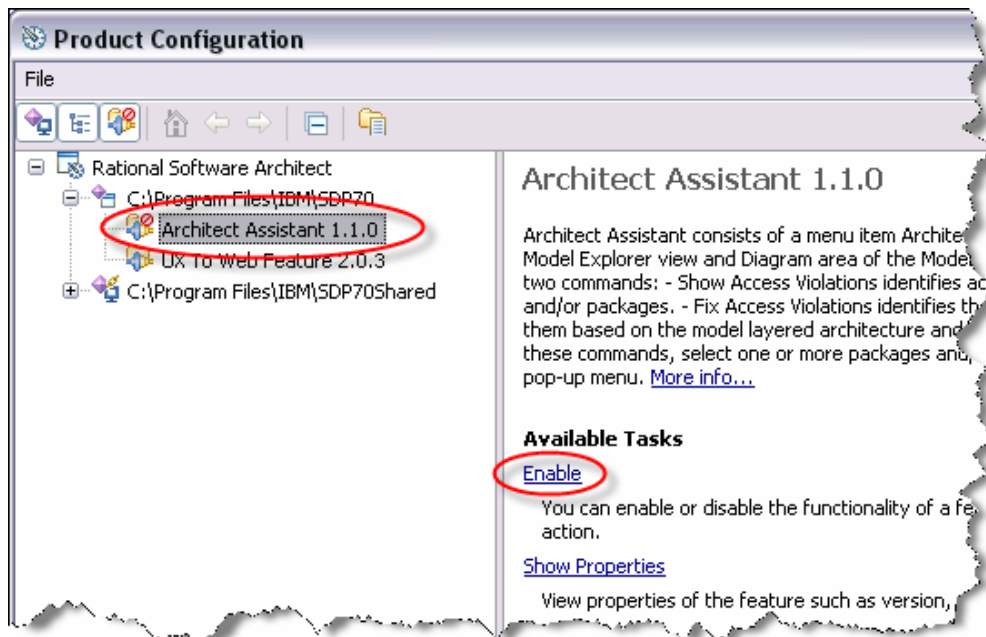
3. If the *ArchitectAssistant* submenu is not available, the plug-in may be installed but not enabled:

- Select *Help > Software Updates > Manage Configuration* (this may take a long time).
- Open the Eclipse location where you installed your plug-in (*C:\Program Files\IBM\SDP70* in our example).
- If you do not see an entry *Architect Assistant 1.1.0*, press the *Show Disabled Features* icon as shown below:



- If you now see the *Architect Assistant 1.1.0* entry with the symbol , the feature is disabled. At this point, you simply need to click on *Enable* as shown below and restart the workspace as requested. If you don't see the *Architect*

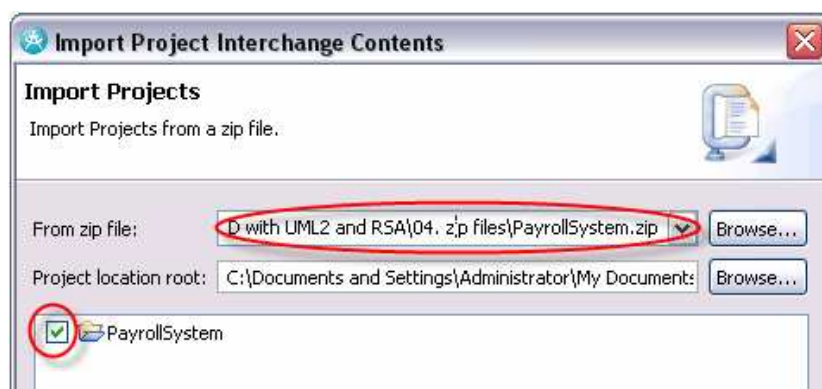
Assistant 1.1.0 entry, it is likely you have not extracted the plug-in to the correct location. Go back to step 1!



4. Remove the *-clean* option from the shortcut (or create a separate shortcut).

Importing the Payroll System solutions (RSA only – for RSM, see next section):

1. *File > Import ... > Project Interchange > Next*
2. In the *Import Project Interchange Contents* dialog:
 - a. Select *PayrollSystem.zip* using the *Browse* button to the right of the field *From zip file*:
 - b. Check the *PayrollSystem* project and click *Finish*

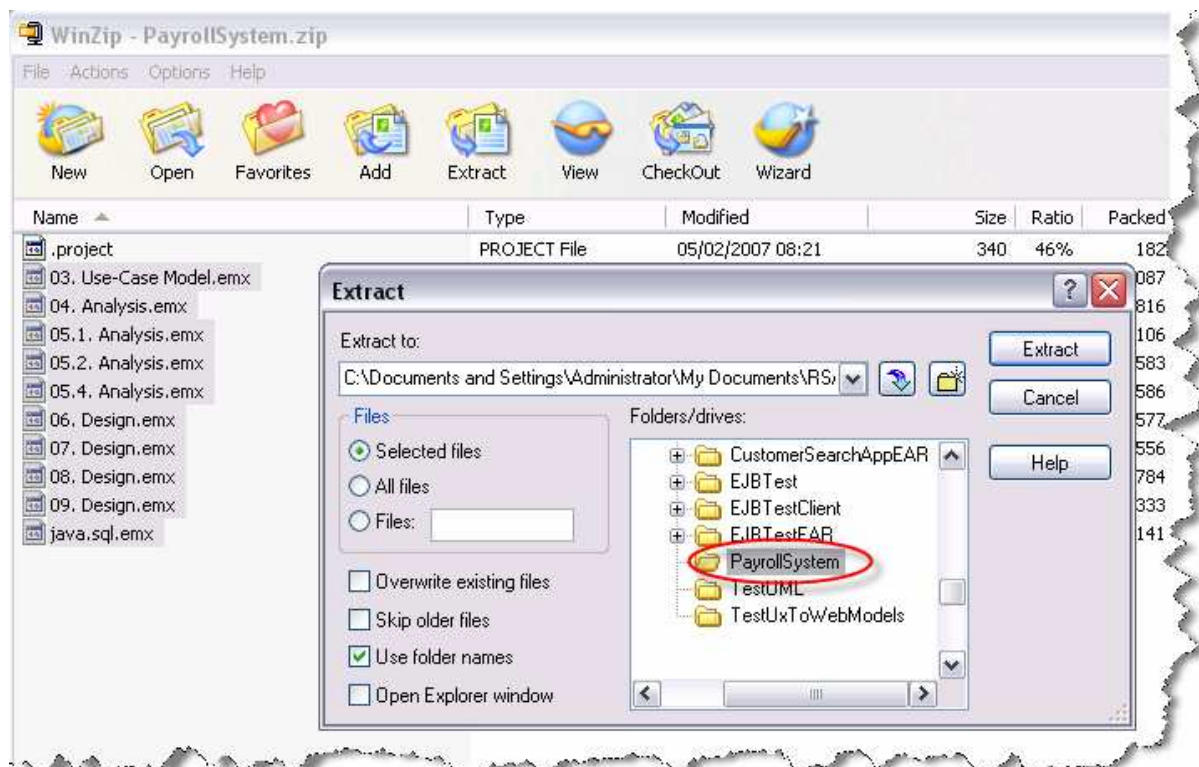


Importing the Payroll System solutions (RSM – also works with RSA):

1. If it is not already the case, start RSM (or RSA) and go to the *Modeling* perspective.
2. Create a new project called *PayrollSystem* using the *Project wizard*:



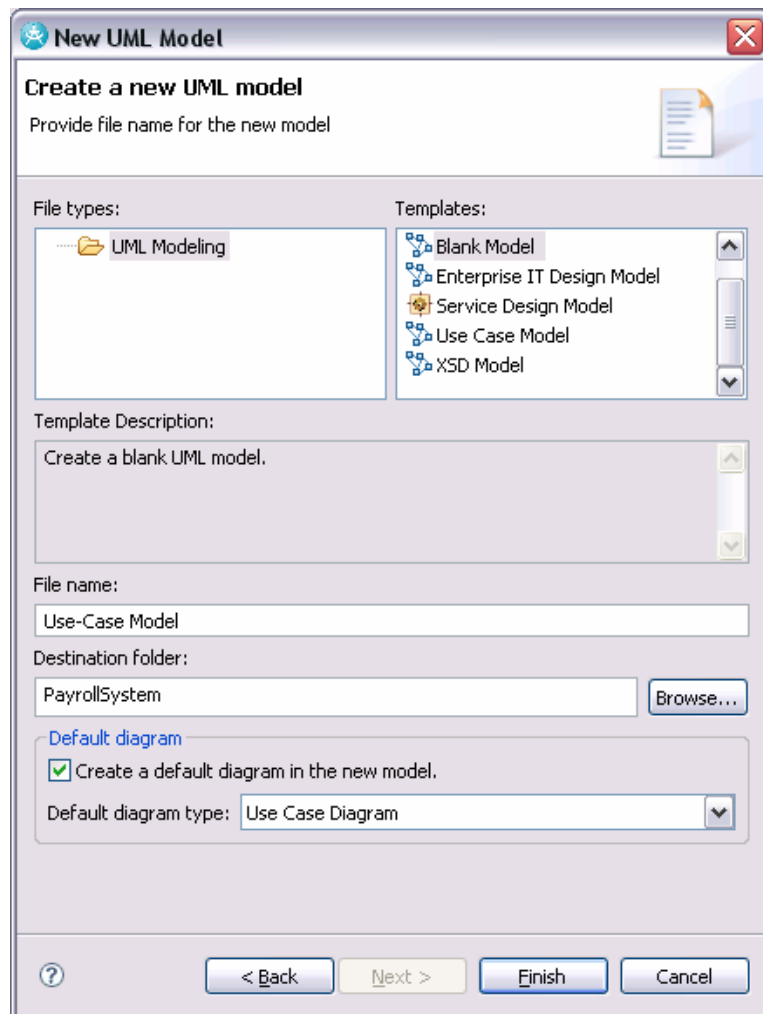
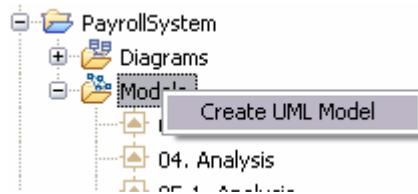
3. Open the zip file, select all the .emx file (see below) and extract them into the directory where your project was created (check out the full path of the project in the *Properties* window):



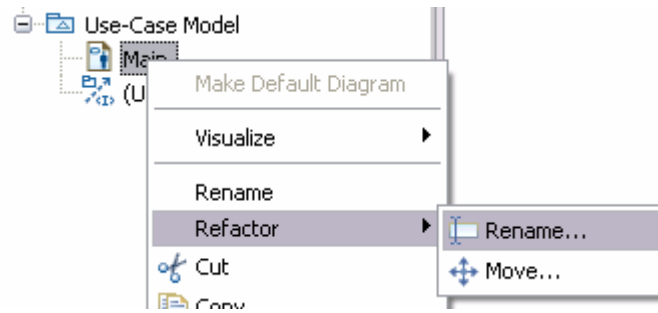
4. In RSM, right-click the *PayrollSystem* project and select the *Refresh* command to see the files in the *Project Explorer*.

Task 3.2: Identify the actors and use cases of the payroll system

1. If it is not already the case, start RSM (or RSA) and go to the *Modeling* perspective.
2. Create in the *PayrollSystem* project a new blank model named *Use-Case Model* with *Use Case Diagram* as the default diagram type.



3. Rename the diagram *Main* as *Global View*:



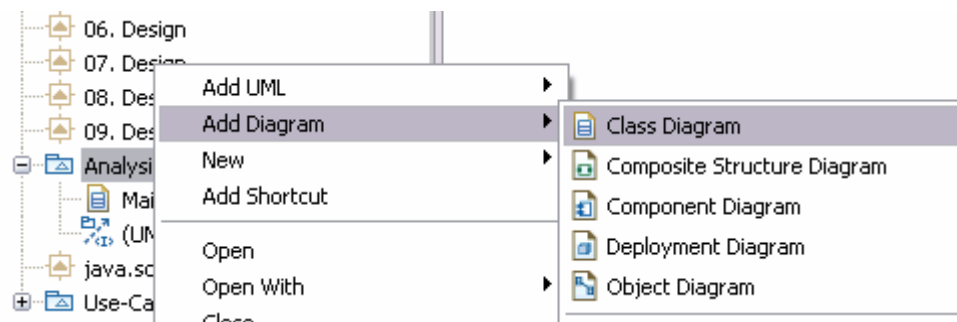
4. Create and briefly describe the actors and use cases for the Payroll System in this diagram, based on the problem statement, glossary and supplementary specifications provided in the Appendix 2 of this document.



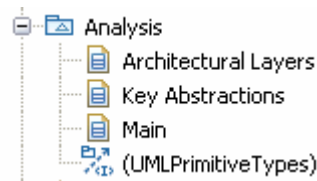
Lab 4 – Architectural Analysis

Task 4.1: Creating the Analysis Model

1. In the *PayrollSystem* project, create a new blank model, named *Analysis*, and with class diagram as the default diagram type.
2. Create in the *Analysis* model two class diagrams respectively names *Architectural Layers* and *Key Abstractions*.

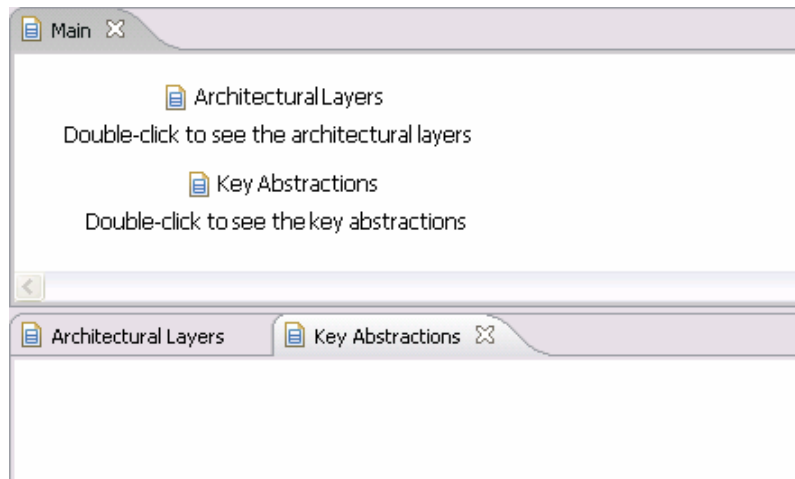


The analysis model should look like this in the project explorer:



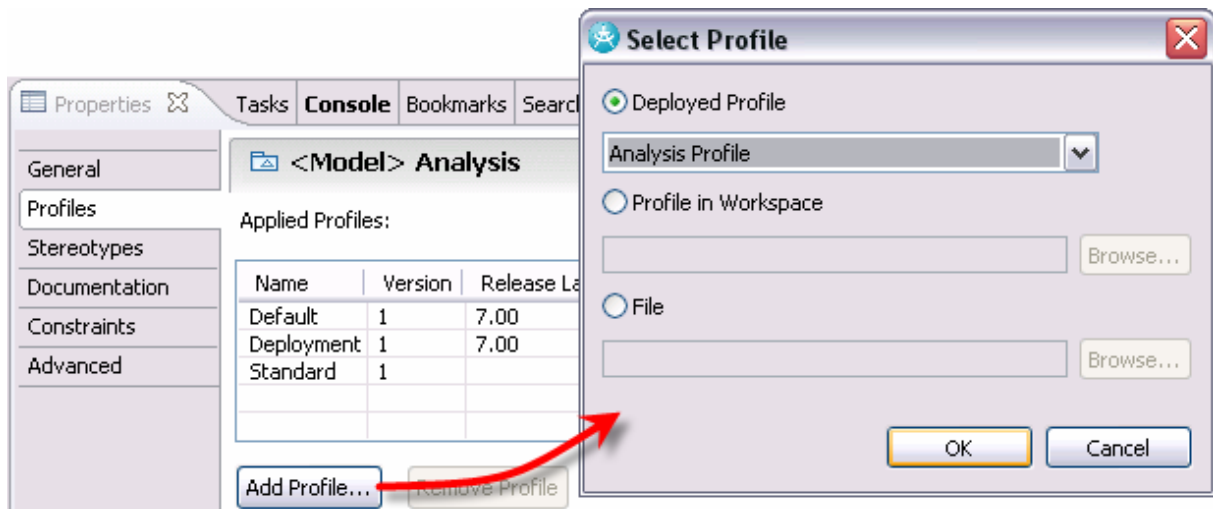
3. In the *Main* diagram, create shortcuts to the newly created diagrams. To do this, you can simply drag and drop each diagram (make sure to read the note below before you do) from the *Project Explorer* onto the *Main* diagram.

Note: In some configurations, clicking on a diagram in the project explorer or attempting to drag-and-drop it automatically brings the diagram to the front if it is already open, thus hiding the intended target (the *Main* diagram). To avoid this situation, close the *Architectural Layers* and *Key Abstractions* diagrams or use “split screens” as in the snapshot below.

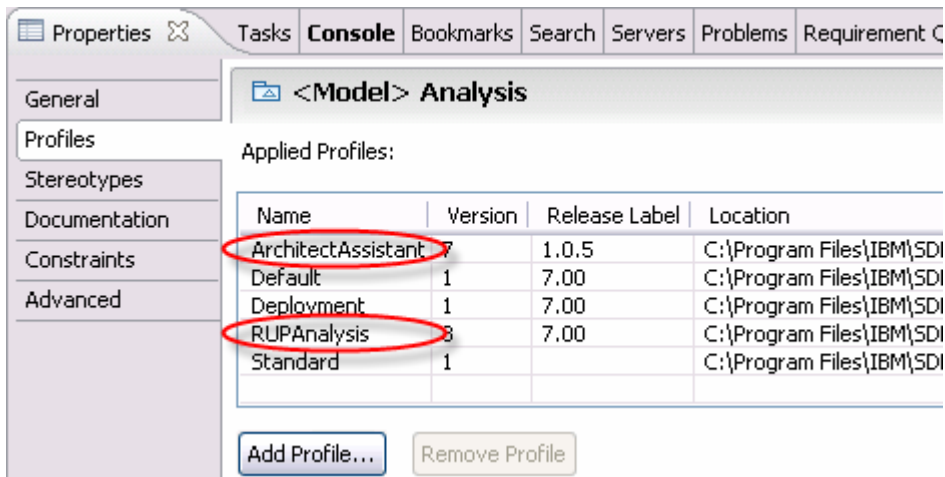


Note: Every package in a model (including the model itself) should have a “default diagram”. This default diagram is the entry point into the package, i.e. the diagram that is opened when double-clicking on the package. The *Main* diagram should contain all relevant information for the user to easily find his/her way in the package (shortcuts to other diagrams, main nested packages, textual information and notes, etc.).

4. Add the profiles *Analysis Profile* and *ArchitectAssistant* to the *Analysis* model. First select the model in the project explorer, then the *Profiles* tab in the *Properties* view. You will need to repeat the *Add Profile* for each profile.

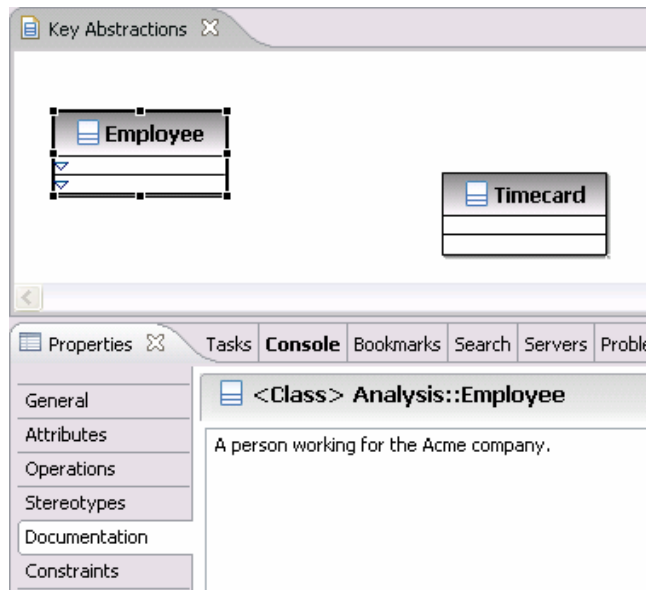


The profiles should appear in the *Applied Profiles* list. We will use them in later labs.

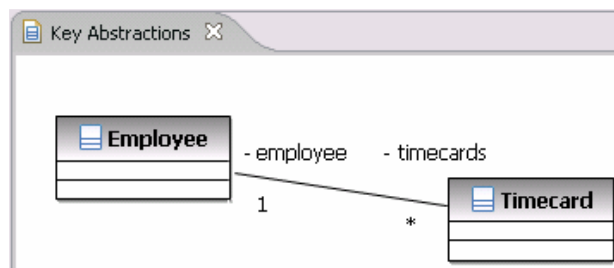


Task 4.2: Identify the Key Abstractions of the Payroll System

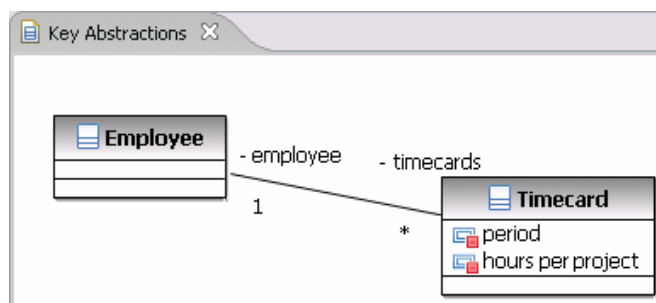
1. For this task, you need:
 - a. The problem statement and glossary for the Payroll System in Appendix 2 You should already be familiar with these documents as you used them in lab 3.
 - b. The Use-Case Model created in lab 3: you can either use the model you created or the solution provided in model 03. *Use Case Model*, which you can now open.
2. Open the *Key Abstractions* diagram in the *Analysis* model.
3. Identify the key abstractions of the system and represent them as classes in the *Key Abstractions* diagram:
 - a. Remember: A key abstraction is a concept, an entity that the system must be able to handle. The key abstractions form an initial set of classes that is useful to “jump-start” the analysis work.
 - b. As an example, consider the following extracted from the problem statement: “*Some employees (...) submit timecards that record the date and number of hours worked for a particular charge number.*”
 - i. Employees and timecards are major entities that the system will have to handle. They are key abstractions. We therefore want to create two classes to represent them: *Employee* and *Timecard*. This is also the right time to provide a brief description of each class.



- ii. Because each employee “owns” the timecards that he/she submits every week, we will add an association between the 2 classes. An employee may have 0 to n timecards. A timecard only makes sense if it can be associated with exactly 1 employee.



- iii. Because it is also said that the timecards record the date and number of hours worked for a particular charge number, we can add *period* (date is not enough) and *hours worked per project* as attributes of *Timecard*.



Should *hours worked per project* be more detailed? First, keep in mind that the purpose of key abstractions is not to create classes that will survive throughout design (although most will). It may make sense to provide a more detailed representation (introducing additional classes), but it is only just that: another representation of the same information... And there are other factors: knowledge of the business domain, whether this is a new application or an overhaul of an existing one, etc.

Note: When a class is created in a diagram, its parent is the package containing the diagram. For now, we will not worry about the exact location of the classes. If it helps you organize the information, feel free to allocate those classes to specific packages, but be aware we may have to change this allocation.

Task 4.3: Represent the Higher Layers of the Architecture

The architect has indicated that, at this stage of the analysis, two architectural layers must be created : the *Presentation* layer and the *Business* layer. The *Presentation* layer depends on the *Business* layer.

Create two packages, *Presentation* and *Business*, in the *Architectural Layers* diagram. Assign the stereotype <<*layer*>> to these packages.

Draw the necessary relationship to support the statement “the *Presentation* layer depends on the *Business* layer”.

Note: For the time being, it is not necessary to assign abstractions to specific layers.



Lab 5 – Use Case Analysis

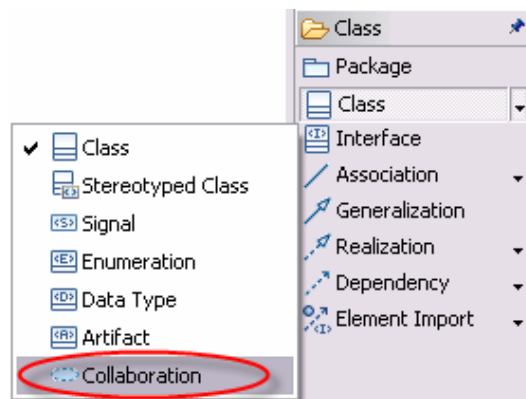
Task 5.1: Create Use-Case Realizations

You have been assigned the use case *Maintain Timecard* to analyze.

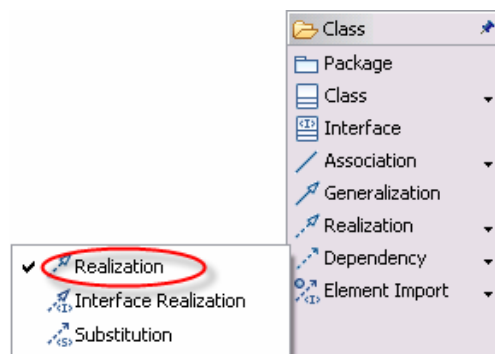
First create the package *UC Realizations* in the analysis model. Add the newly created diagram to the *Main* diagram of the model.

In the package *UC Realizations*:

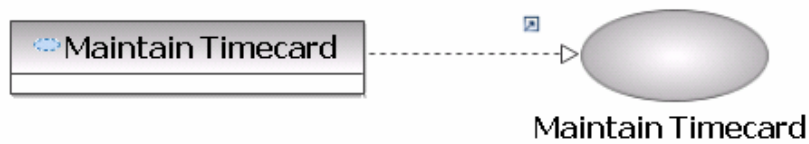
1. Create a package called *Maintain Timecard* and add this package to the *Main* diagram of the *UC Realizations* package.
2. In the *Main* diagram of the package *Maintain Timecard*, create a use-case realization named *Maintain Timecard*.
 - a. Use the *Collaboration* tool in the *Class* drawer under the *Class* entry



- b. Still in the same diagram, add the use case *Maintain Timecard* created in the lab 3 (or use the use case from model 03. *Use-Case Model*).
 - c. Draw a *Realization* relationship from the use-case realization to the use case.



The diagram should look something like:



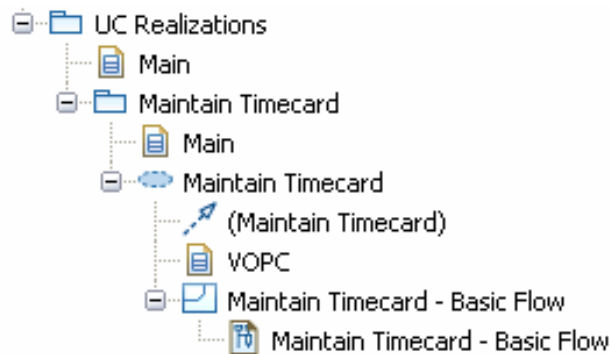
Task 5.2: Assign the Use-Case Behavior to Classes

For this exercise, you will need the specification for the *Maintain Timecard* UC in appendix 2 of this document. Also, you can either work from the *Analysis* model created during the previous exercise, or from model *05.1. Analysis*.

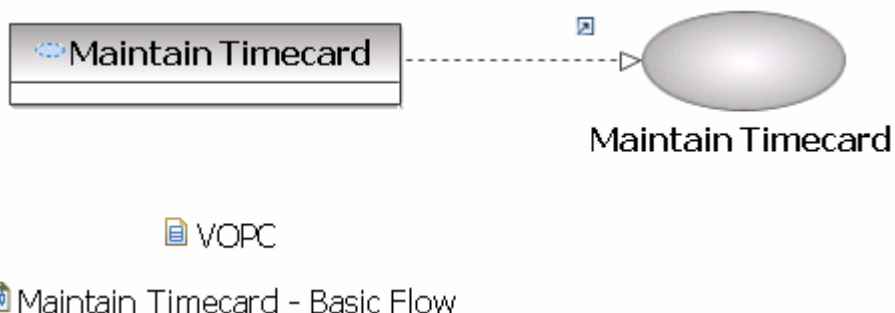
If this is not already the case, make sure the *Main* diagram for the *Maintain Timecard* UC Realization is displayed on your screen.

Create a new sequence diagram for the basic flow of the use case:

1. In the *Project Explorer*, right-click the *Maintain Timecard* collaboration (☁) and select *Add Diagram > Sequence Diagram*.
2. Name the interaction (📄) and the diagram (📄) *Maintain Timecard – Basic Flow* (see below).
3. Create a class diagram called *VOPC* under the *Maintain Timecard* collaboration (see below)



4. Drag and drop the *VOPC* and *Maintain Timecard – Basic Flow* diagrams onto the *Main* diagram of the *Maintain Timecard* UC, which should look like this:



5. From the textual description of the basic flow, identify the classes that participate in the use case and assign use case behavior to these classes as this was previously shown by the instructor:
 - a. Each object must be assigned to a class (existing or new).
 - b. Each message between objects must correspond to a class operation.
 - c. Each participating class will be added to the *VOPC* diagram (note: for the time being, we are not concerned with the actual location of the classes – this will be dealt with at a later time).
 - d. For each class, define its documentation, analysis stereotype (*boundary*, *control* or *entity*), and main attributes (*entity* classes).

Task 5.3: Complete the VOPC

For this exercise, you will need the *Maintain Timecard – Basic Flow* sequence diagram created in task 5.2. You can either work from the *Analysis* model created during the previous exercise, or from model 05.2. *Analysis*.

Open the *VOPC* diagram for the *Maintain Timecard* UC Realization.

Complete the diagram as follows:

1. Add stereotypes to all classes (if not already done).
2. Create relationships between classes: remember, every link between object is an instance of a relationship between the corresponding classes:
 - e. Specify multiplicity and role names as appropriate.
3. Add class responsibilities (if not already done).
4. Add relevant attributes to entity classes (if not already done).

Task 5.4: Map Analysis Mechanisms to Classes

The architect has indicated that the following analysis mechanisms have been identified:

1. Persistence
2. Distribution
3. Security
4. Legacy interface

This exercise will be done as a group discussion.



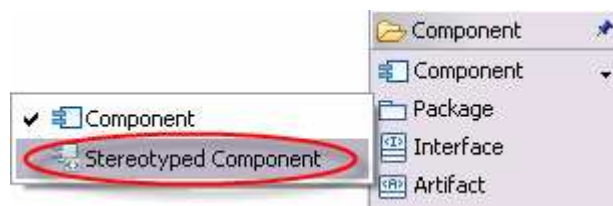
Lab 6 – Identify Design Elements

Task 6.1: Transform an Analysis Class into a Subsystem

For this exercise, you will work from the model 05.4. *Design*.

Your task is to transform the *ProjectManagementDB* into a subsystem + corresponding interface:

1. Create the subsystem *ProjectManagementDB* in the *Business* package:
 - a. Method 1: right-click the *Business* package and select *Add UML > Subsystem*.
 - b. Method 2: in the drawing area, use the *Stereotyped Component* tool in the *Component* drawer (see below), and select the *Create <<subsystem>> component* menu entry.



- c. Name the component *ProjectManagementDB* (be careful not to delete the stereotype while renaming it).
2. Create a package *ProjectManagementData* in the *Business* package. This new package will contain the interface corresponding to the subsystem as well as associated data.
3. In the *ProjectManagementDB* subsystem, create a class diagram called *Subsystem ProjectManagementDB*:
 - a. Right-click the subsystem in the *Project Explorer* and select *Add Diagram > Class Diagram*.
 - b. Name the diagram *Subsystem ProjectManagementDB*.
 - c. Right-click the newly created diagram in the *Project Explorer* and select *Make Default Diagram*.
4. Using the examples provided in the section *Identify Subsystems and Subsystem Interfaces* from module 10, create the interface *IProjectManagementDB*:
 - a. Move the boundary class *ProjectManagementDB* into the subsystem (and remove its stereotype).
 - b. Transform the responsibility // *retrieve charge numbers* in a full operation with the appropriate parameters and return value.
 - c. Create the interface *IProjectManagementDB* (in the package *ProjectManagementData*), define its operations and add realization relationships from the subsystem *ProjectManagementDB* AND the class *ProjectManagementDB* to this interface.

5. Replace all references to the analysis class *ProjectManagementDB* by references to *IProjectManagementDB* in the rest of the model (hint: check the *Maintain Timecard* UC Realization).



Lab 7 – Class Design

For this exercise, you will work from the model *06. Design*.

Inspect the model:

1. Open the model if not already done.
2. In the *Main* diagram, double-click on the *Business* package, then on the *Employee*. This diagram contains the class *Employee*, its subclasses and several associated classes:
 - a. The class *Paycheck* is not displayed not to overload the diagram.
 - b. Note the attributes in the *Employee* class: they are now all typed except the *bank info* attribute for which we didn't have enough information.
 - c. Still in the *Employee* class, note that most operations are shown with their I/O parameters and return values. For the operation *getMethodPayment()*, we have introduced a new class of type *Enumeration*.
 - d. In the class *PurchaseOrder*, getters have been added.
 - e. Finally, note that the compositions *Employee > Timecard*, and *CommissionedEmployee > PurchaseOrder* are unidirectional.
3. Note: In order to type the miscellaneous attributes and parameters, the classes *Date* and *Double* have been added to the *Primitives* package.

You have been asked to complete the following points in the diagram:

1. For the remainder of the exercise, use ONLY the types already defined (including *Date* and *Double*):
 - a. A time interval or period (of time) will be expressed with two attributes (or parameters) of type *Date*, for instance *from : Date* and *to : Date*.
2. Change the responsibility // *retrieve amount to pay* in the *Employee* class into a *calculatePay()* operation:
 - a. *calculatePay()* returns the amount to pay an employee for a given period of time.
 - b. It must be designed with polymorphism in mind: it must be possible to calculate the amount to pay for an *Employee* object WITHOUT knowing its type (hourly, salaried, commissioned).
3. Update the analysis class *Timecard*:
 - a. Consider in particular the representation of the analysis attribute *hours per project*.
4. A new requirement has been added: it must be possible to change an hourly employee into a salaried employee. What do you suggest to handle this situation? (Discussion with the instructor.)



Lab 8 – Subsystem Design

For this exercise, you will work from the model *07. Design*.

Your task is to apply the JDBC mechanism described in module 13 to the subsystem *ProjectManagementDB*:

1. In the *ProjectManagementDB* subsystem, create a collaboration named *IProjectManagementDB Implementation*.
2. In the collaboration, create a *VOPC* class diagram and a sequence diagram titled *retrieveChargeNumbers Implementation*.
3. Apply the JDBC mechanism:
 - a. Complete the subsystem class diagrams based on the JDBC mechanism described in page 44 or 88 of the Part III – Object-Oriented Design fascicule. Note: the *java.sql* classes are available in the model *java.sql*.
 - b. Optionally, build the sequence diagram for *retrieveChargeNumbers Implementation* based on the diagram on page 46 of the Part III – Object-Oriented Design fascicule.



Lab 9 – Finalize the Design Model

For this exercise, you will work from the model *08. Design*.

All the classes you have created must be assigned to packages. The packages in turn must be assigned to the layers of your architecture. Relations between classes determine the dependencies between packages. From the resulting hierarchy depend the potential for reuse, scalability and flexibility of your system.

Allocate all classes to packages contained in the *Presentation* and *Business* layers. You can either build your own hierarchy from scratch, or use any of the following suggestions (note: these suggestions are minimalist – in real life, the package structure will be significantly different):

1. Regroup the control classes, *TimecardForm* and *ServiceScheduler* in two packages in the *Presentation* layer (for instance *AdminActivities* and *EmployeeActivities*).
2. All other classes are allocated to the *Business* layer:
 - a. Classes associated with the employee are stored in a package *EmployeeData*.
 - b. The other classes *BankSystem* and *PrintingService* are stored in a package *OtherServices* (this is an extreme simplification that would never survive in a real system).
3. Stereotype the package *UC Realizations* as <<perspective>> (this means that *UC Realizations* only contains diagrams and no classes or other UML elements).
4. Add the stereotypes *global* and *layer* to the package *PrimitivesTypes* (again reality would be quite different but it does not modify our messages).

Solve the access violations:

1. Right-click on the *Design* model in the *Project Explorer*, then select *ArchitectAssistant* > *Fix Access Violations: Fix Access Violations* attempts to automatically resolve the access violations based on the layered architecture.
2. When a potential access violation is identified, *Fix Access Violations* can:
 - a. Create a dependency if the violation can be solved automatically.
 - b. Report an error in the case of a “true” violation (for instance if the *Business* layer attempts to access the *Presentation* layer).
 - c. Prompt the user in all other cases.
3. Inspect the *Console* and/or *Problems* view to identify the remaining access violations and modify the model to eliminate them.

Create a diagram titled *Package Dependencies* in the *Design* model (at the same level as *Main*, *Key Abstractions*, and *Architectural Layers* diagrams). Add to this diagram all the packages and subsystems from the layers. Inspect the result.

Appendix 1 – Course Registration Requirements

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 the 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 Specifications

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 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.

Register for Courses UC

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.

Appendix 2 – Payroll System

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 Specifications

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.).

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
The system shall provide a Windows-based desktop interface.

Maintain Timecard UC

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.

Run Payroll UC

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.