Manage Project Homework: Design Document

Riccardo Ancona - 782025
Alessandro Ditta - 781482

January 24, 2012

Revision Summary

1. Date: 17/01/2012

   Revision: 1

   Changes:

   Added details to the classes present in the Connection package of the Client Module.

   Reviewed the GUI classes.

2. Date: 18/01/2012

   Revision: 2

   Changes:

   Added details to the Local package in the Server Module.

   Rewritten Section no. 2.

   Added some additional pseudocode examples.

   Introduced list of figures.

   Added JUnit as external tool.

3. Date: 22/01/2012

   Revision: 3

   Changes:

   Reviewed the Client Module description.

Contents

List of Figures

1 Introduction

1.1 Purpose

This software design document describes the architecture and system design of Manage Project Homework

It is a living document that is expected to evolve throughout the design process. The focus during conceptual design is of describing enough of the design to allow an examination of the design's suitability in meeting the system requirements contained in the Requirements Analysis and Specification Requirements document.

The intended audience of this document includes the prospective developers of the tool.

1.2 Scope

The software system to be produced is a projects management tool which will be referred to as Manage Project Homework (MPH) throughout this document.

MPH will allow professors to publish a project description and to define the set of the corresponding deliverables. It will allow students to join project teams and submit deliverables by uploading them into the system.

The professor will also be able to evaluate the project deliverables assigning a score to them and leaving to the system the computation of the final score based on the average of the individual scores. MPH will also provide some information sharing functionalities among different teams.

1.3 Definitions, acronyms, and abbreviations

MPH: Manage Project Homework, the software system to be produced.

Team: a set of 1, 2 or 3 students who work together on the same project.

Admin: the system administrator

Deliverable: the model of a tangible object produced as a result of a specific phase of the project.

Artifact: the effectively tangible object associated to a deliverable.

Deadline: the date by which the artifact associated to a deliverable must be delivered.

RASD: Requirements Analysis and Specification Document

DD: Design Document

DBMS: DataBase Management System

GUI: Graphical User Interface

RMI: Remote Method Invocation

JEE: Java Enterprise Edition

UX Diagram: User Experience Diagram

DTO: Data Transfer Object

MVC: Model-View-Controller

1.4 References

- Description of the project: *http://corsi.metid.polimi.it*
- MPH RASD
- The Interactive User Experience (UX.html) dynamic web page attached to this document
- IEEE Std. 1016-2009: IEEE Standard for Information Technology—Systems Design— Software Design Descriptions: *http://ieeexplore...number=5167253*
- Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli, Ingegneria del Software - Fondamenti e Prinicipi, Pearson Education Italia, 2004

2 Functional overview of modules and components

The MPH system to be developed consists in a *Client-Server* application with *fully remote presentation.* The software is integrated with a database accessible by the Internet. The design process takes great care in designing a framework which can be updated easily based on a multi-tier architecture. There are three basic, logical components of the system: the Database Management System (Data Tier), the Server Module (Business Tier), and the Client Module (Presentation Tier). The system architecture follows the Java Enterprise Edition (JEE) paradigm.

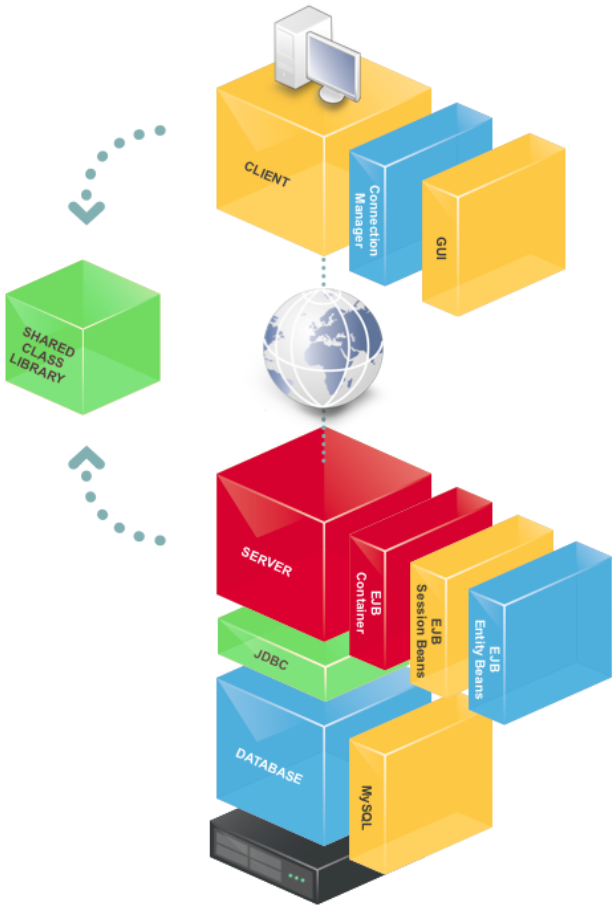

Figure 1: Architecture Logic Tiers

The software will be deployed in two executable jars, one for the client, and one for the server. The **JBOSS** application server and the **MySQL DMBS** are needed to run on the server host.

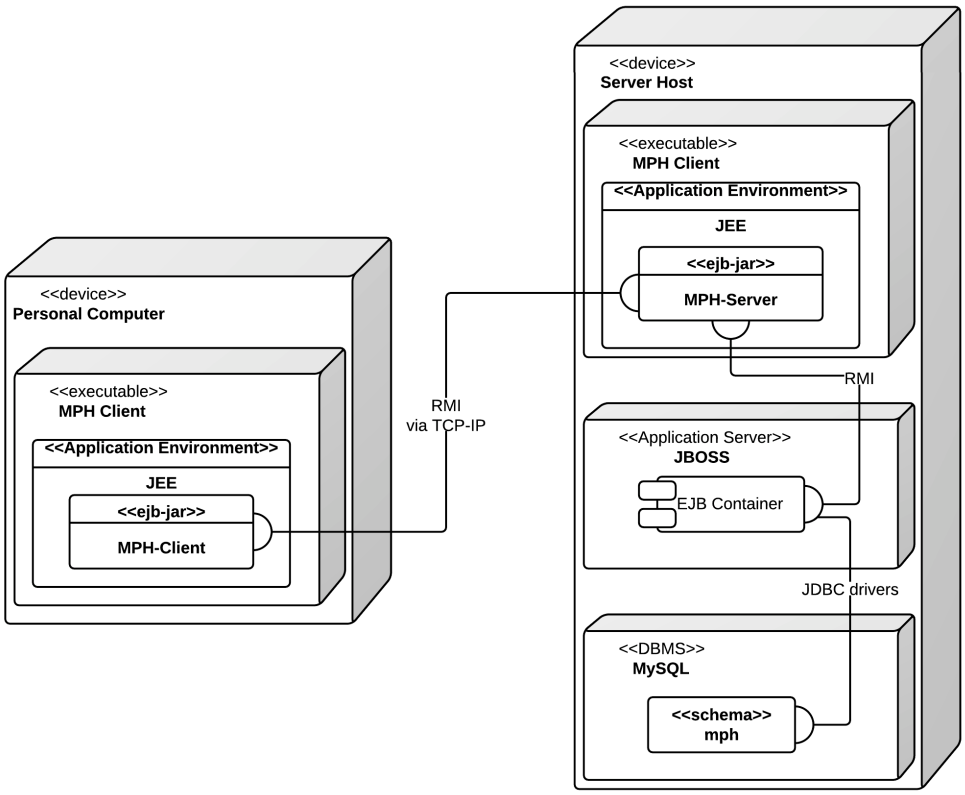

Figure 2: MPH Deployment diagram

The general modules of the software are now presented in a basic overview. They will be presented in a more detailed view in the Detailed System Architecture Description section.

2.1 Database Management System

The database hosts the back-end database which is used for central data storage. It will contain tables to represent the project data. We will be using MySQL COTS as our database software. The database is mapped to particular JEE objects called entities: in this way, JEE framework will take care of its maintenance and update, virtually hiding the underlaying structure, letting the programmer to manage it in a object-oriented way.

2.2 Server Module

The server software will directly manipulate the contents of the database, based on commands received from the Client. It will be implemented in Java, using

the JEE framework. It contains an EJB Container, where the business logic is executed through EJB Session Beans, using the data collected from the database using EJB Entity Beans.

It is structured in two layers: the *remote layer* and the *local layer*.

The remote layer is made by stateful session beans that will accept commands from the client module: each command is translated into several commands executed locally by the local layer. In this way, all the session beans of the remote layer implements the design pattern *Facade*.

The local layer is made by stateless session beans that will accept commands from the remote layer: each command create or edit entities. In this way, all the stateless session beans of the local layer implements the JEE design pattern *Data Access Object* (you can find more information about this pattern at thislink).

If the client module requests some data, instead of passing back an entity, a data transfer object is passed. For more information, please see the Data Transfer Objects package of the Shared Class Library.

2.3 Client Module

The Client Software will reside on a desktop computer. It will be implemented in Java, using Swing java libraries. Its purpose is to present data to the user as requested, and provide an interface so that the user can easily update project information and other data fetched from the server. In order to do this, it uses the Shared Class Library Module that contains all the server interfaces that the client could invoke through Remote Method Invocation.

The Client will make use of the Singleton design pattern, in fact all the 3 classes contained in the Connection package (ConnectionManager, ProfessorManager, StudentManager) are Singletons. This is useful when exactly one object is needed to coordinate actions between the Client and Server.

The Client will also implement the MVC architecture pattern: user commands are taken by the GUI (View) which invokes the corresponding methods of the ProfessorManager or StudentManager (Controller), they are responsible for throwing exception which will be caught by the GUI and shown to the user. Data to send to the Server or to be displayed to the user are stored in DTO's.

2.4 Shared Class Library Module

The shared class library will provide some common interfaces for the client and server module, as well as some common useful utility classes. It provides a uniform representation on both sides of the network connection, providing local methods for invoking remote methods on the server application hiding the details of how communication with the server is achieved. Moreover, client application can use data transfer object defined in shared class library to obtain data from the database without knowing the implementation of each java bean entity: in this way, we can totally decouple the client from the server hiding the complexity of the internal business logic and the complexity the internal

data organization. More info will be provided in the Data Transfer Objects and Sessions sections.

The business logic of the server is exposed to the client showing interfaces that represent all the functionalities of the actors and of the mph system that were extracted from the RASD document.

Each actor is defined as an interface that implements all the functionalities assigned to that actor.

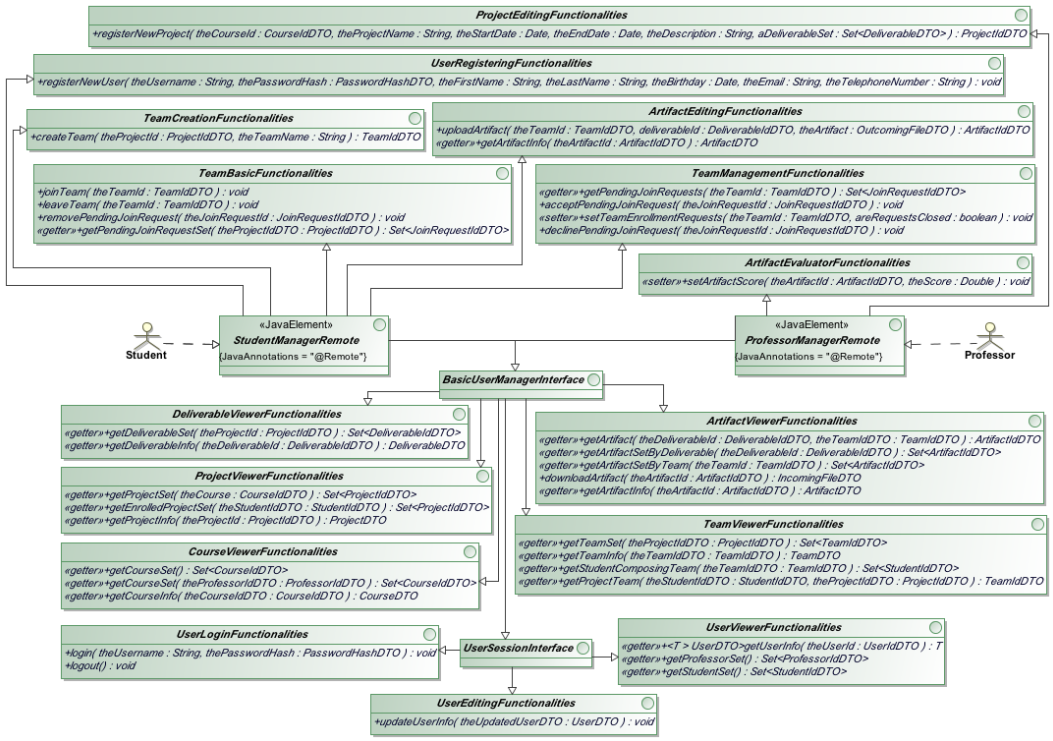

Figure 3: Actors functional view

Both student and professors, implements some common functionalities, contained in the BasicUserManagerInterface, such as Login Functionalities, User Profile Editing Functionalities, Team Viewing Functionalities etc. In general, all the shared functionalities between students and professors are "read-only" functionalities. "Write-only" functionalities are separated for students and professors: students implements the ability to create and manage teams, to register themselves into the mph system and the ability to upload artifacts (each functionality is embedded in the proper interface), while professors implements the ability to evaluate artifacts and to edit and create new projects. All the student's functionalities are contained in the StudentManagerRemote, while the

professor's one are contained in the ProfessorManagerRemote. This two interfaces are exposed both to the server and to the client, so the client is able to know what actions can perform, and the server is able to know what action can expect to be requested from the client.

The extreme fine granularity of the defined functionalities, even if not requested by the requirements, could let in the future to create more easily new user typologies, for example professor that cannot mark artifacts or student that cannot manage a team, thus it is a precise design choice to make the mph software more modular and flexible.

To let the server pass data to the client without exposing the underlaying structure of the server, the JEE design pattern *Data Transfer Object* is used: each entity of the server is mapped by a specific data transfer object, that contain all the main attributes that the client needs.

2.5 Systemic view

After detailing all the modules, here we are a systemic view.

The client module establish a connection with the server module through the stateful session beans. Each request of the client is decomposed in multiple requests sent to stateless session beans, that have access to the underlying database and modifies or creates new data. After a request is executed, if the request expected a return value, the request creates a Data Transfer Object and return it to the client. The stateless session beans use the Util package of the shared module and can throw exceptions that are catched by the client, that shows them to the user with the GUI .

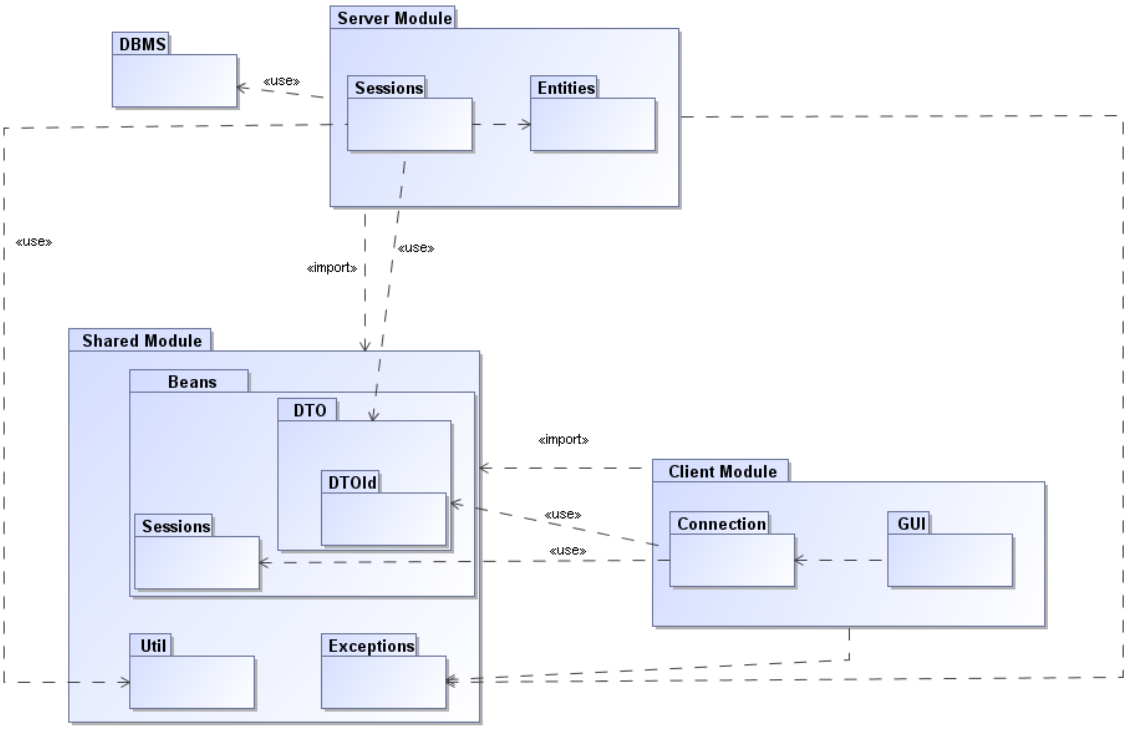

Figure 4: MPH general architecture

The following tables will provide more details on each module and component.

3 Detailed System Architecture Description

3.1 Database Management System

Identification	Database Management System Software
Type	Module
Purpose	Provides data management and storage for the server module.
Function	Takes EJQL/SQL QUERY and UPDATE commands from the server module, and stores or updates the data according to those commands.
Subordinates	To see the subordinates data structure, check the following E-R diagram.
Dependencies	A system administrator user must perform database setup functions, adding and modifying the structure of tables so that the server module can store the data as appropriate.
Interfaces	Apart from administrative setup, all modification of the database items will be performed via JDBC by the server application, exploiting the mapping between Entity Beans and Database Tables.
Resources	The database that will be used is MySQL 5.0. All tables will be created by the software at run-time. The software module will connect to the database using JDBC connectivity. Once the server gets a request for a certain data type from a client, the server will connect to the database, execute the query and get a java ResultSet back. At this point, it will convert the ResultSet to the Object that was requested by the client and send it back to the client via RMI over the network. Calls to the database should not take longer than (but should be much less than) 30 seconds and transfers of the each Object over the network should not take longer than 1 minute. Conversion of a ResultSet to a Java Object should take less than 1 second. A client may also send an updated object to the server, in this case it will convert to a Set and then do an UPDATE to the database. This is a basic database call so it should not take longer than 1 second from start to finish.
Processing	When the server module execute a request for the client, it will query the database for some information, then will perform some kind of operation and possibly modify the database state. For an example you can go and check a sample pseudocode (see A.1) in the appendix of this document.
Data	The data in the database will be filled by the server module except for the cases stated above (during extraordinary maintenance by the system administrator). We will use valid EJQL or SQL statements called within Java through JDBC to modify and update the data. In every case we will either get a ResultSet (for queries), a working return code or a SQLException if the command does not work for any reason. Exceptions will be handled according to their type.

Table 1: Database Management System detailed info table

3.1.1 Conceptual Entity-Relationship Diagram

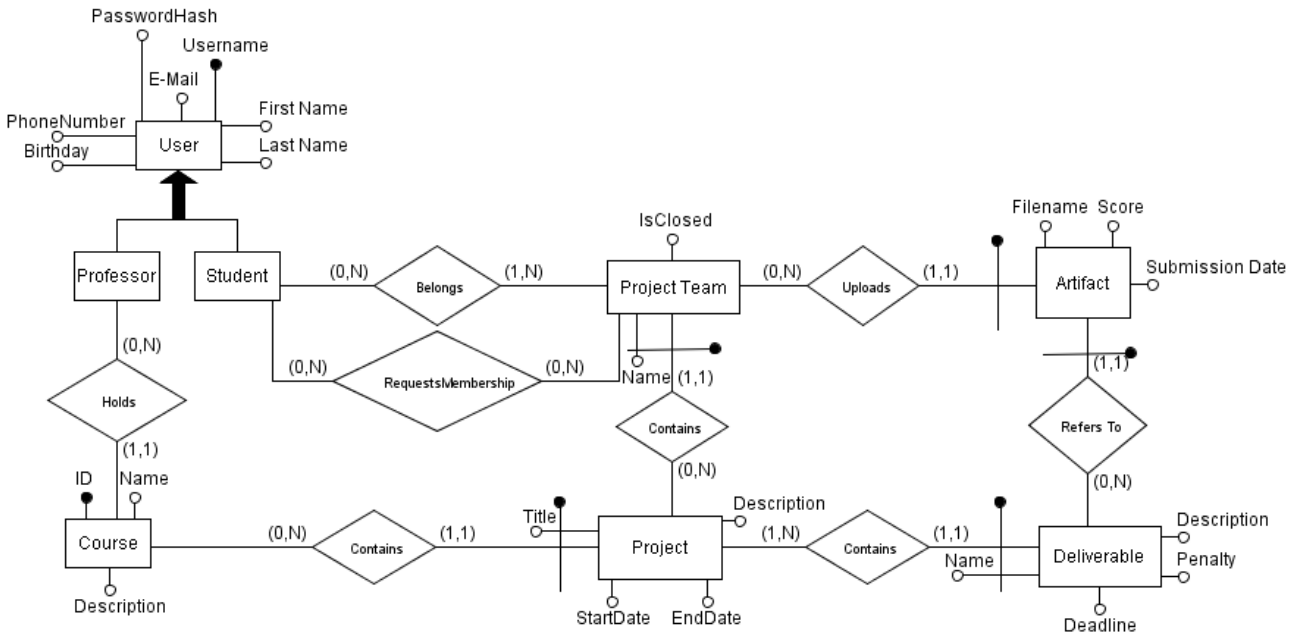

Figure 5: Conceptual Entity-Relationship Diagram

3.1.2 Logic Entity-Relationship Diagram

3.1.2.1 Operations The *User* generalization was removed by sub-classing it into the tables *Professor* and *Student*: although they shared the same data, they were deeply different respect to other relationships throughout the scheme, and to remark this difference, it was decided to keep them separated. Every weak entity, such as *Project Team*, *Project*, *Deliverable* and *Artifact* are modeled as a standalone entity that contains its own primary key and the primary keys of the entities on which they depend. The many-to-many *Belongs* relationship was resolved by creating a bridge table *StudentBelongsTeam* containing the primary keys of both the entities that joined the relationship. Other operations are self-explanatory.

3.1.2.2 Tables

- **Professor**(Username, PasswordHash, FirstName, LastName, Birthday, E-Mail, PhoneNumber)
- **Student**(Username, PasswordHash, FirstName, LastName, Birthday, E-Mail, PhoneNumber)

- **Course**(ID, Name, ProfessorUsername, Description)
- **Project**(CourseID, Title, StartDate, EndDate, Description)
- **Deliverable**(CourseID, ProjectTitle, Name, Deadline, Description, Penalty)
- **ProjectTeam**(CourseID, ProjectTitle, Name, IsClosed)
- **StudentBelongsTeam**(StudentUsername, TeamName, ProjectTitle, CourseID)
- **StudentRequestsMembership**(StudentUsername, TeamName, ProjectTitle, CourseID)
- **Artifact**(CourseID, ProjectTitle, ProjectTeamName, DeliverableName, FileName, SubmissionDate, Score)

3.1.2.3 Foreign Keys

- **Course**.*ProfessorUsername* → ***Professor***.*Username*
- **Project**.*CourseID*→**Course**.*ID*
- **Deliverable**.(*CourseID*, *ProjectTitle*) → **Project**.(*CourseID*, *Title*)
- **ProjectTeam**.(*CourseID*, *ProjectTitle*) → *Project*.(*CourseID*, *Title*)
- **StudentBelongsTeam**.*StudentUsername* → **Student**.*Username*
- **StudentBelongsTeam**.(*CourseID*, *ProjectTitle*, *TeamName*) → **ProjectTeam**.(*CourseID*, *ProjectTitle*, *Name*)
- **StudentRequestsMembership**.*StudentUsername* → **Student**.*Username*
- **StudentRequestsMembership**.(*CourseID*, *ProjectTitle*, *TeamName*) → **ProjectTeam**.(*CourseID*, *ProjectTitle*, *Name*)
- **Artifact**.(*CourseID*, *ProjectTitle*, *TeamName*) → **ProjectTeam**(*CourseID*, *ProjectTitle*, *Name*)
- **Artifact**.(*CourseID*, *ProjectTitle*, *DeliverableName*) → **Deliverable**(*CourseID*, *ProjectTitle*, *Name*)

3.1.3 Detailed Table Description

3.1.3.1 Professor This table contains the *Professors* registered into the MPH system. Every professor is identified by a username. The *PasswordHash* attribute must contain the MD5 hash digest for the real password of the user. The other attributes represent the personal data of the professor.

- **Constraints**
  - *NOT NULL Attributes*

        * *Username*
        * *Password*
        * *E-Mail*
    - *Birthday* must be a *Date* posterior the 01/01/1900 and anterior the actual Timestamp

- **Deleting Policy**
    - Cascade

- **Updating Policy**
    - Cascade

3.1.3.2 Student This table contains the *Students* registered into the MPH system. Every student is identified by a username. The *PasswordHash* attribute must contain the MD5 hash digest for the real password of the user. The other attributes represent the personal data of the student.

- **Constraints**
    - *NOT NULL Attributes*
        * *Username*
        * *Password*
        * *E-Mail*
    - *Birthday* must be a *Date* posterior the 01/01/1900 and anterior the actual Timestamp

- **Deleting Policy**
    - Cascade

- **Updating Policy**
    - Cascade

3.1.3.3 Course This table contains the *Courses* registered into the MPH system. Every course is identified by a unique ID and contains the username of the professor holding the course. It also has got a name and a description.

- **Constraints**
    - *NOT NULL Attributes*
        * *ID*
        * *ProfessorUsername*

- **Deleting Policy**
  - Cascade
- **Updating Policy**
  - Cascade

3.1.3.4 Project This table contains the *Projects* registered into the MPH system. Every course is identified by the corresponding course ID and a title. It contains a description as well as a start and an end date and flag which, if set to true, means the project is closed.

- **Constraints**
- *NOT NULL Attributes*
  - *CourseID*
  - *Title*
  - *StartDate*
  - *EndDate*
- **Deleting Policy**
  - Cascade
- **Updating Policy**
  - Cascade

3.1.3.5 Deliverable This table contains the *Deliverables* associated with a project. Every deliverable is identified by the title of the associated project, the course ID and the name. Every deliverable has also got a deadline, a description and a penalty.

- **Constraints**
  - *NOT NULL Attributes*
    - *CourseID*
    - *ProjectTitle*
    - *Name*
    - *Deadline*
    - *Penalty*
- **Deleting Policy**
  - Cascade
- **Updating Policy**
  - Cascade

3.1.3.6 ProjectTeam This table contains the *Project Teams* enrolled in a project. Every team is identified by the title of the enrolled project, the course ID and the team name.

- **Constraints**
  - *NOT NULL Attributes*
    - *CourseID*
    - *ProjectTitle*
    - *Name*
- **Deleting Policy**
  - Cascade
- **Updating Policy**
  - Cascade

3.1.3.7 StudentBelongsTeam This table is a bridge between the Student table and the ProjectTeam table, it associates a team with every student belonging to it.

- **Constraints**
  - *NOT NULL Attributes*
    - *StudentUsername*
    - *TeamName*
    - *CourseID*
    - *ProjectTitle*
- **Deleting Policy**
  - Cascade
- **Updating Policy**
  - Cascade

3.1.3.8 StudentRequestsMembership This table is a bridge between the Student table and the ProjectTeam table, it associates a team with every student requesting to join it.

- **Constraints**
  - *NOT NULL Attributes*
    - *StudentUsername*

        * *TeamName*
        * *CourseID*
        * *ProjectTitle*

- **Deleting Policy**
    - Cascade
- **Updating Policy**
    - Cascade

3.1.3.9 Artifact This table contains the *Artifacts* delivered by the teams for a project deliverable. Every artifact is identified by the project team key it and the project deliverable key. It also contains the name of the uploaded file, the submission date and the score given by the professor.

- **Constraints**
    - *NOT NULL Attributes*
        * *CourseID*
        * *ProjectTitle*
        * *ProjectTeamName*
        * *DeliverableName*
        * *FileName*
        * *SubmissionDate*
- *Score* is an integer between 1 and 10
- **Deleting Policy**
    - Cascade
- **Updating Policy**
    - Cascade

3.2 Database Object/Relational Mapping

3.2.1 Entities' attributes name modifications

Since we will map the relational database to plain old java objects using Java Persistence Annotation, it is necessary to specify the modifications that our logic entity-relationship diagram will have once mapped to pojos. In general, some attribute will lose the name specified in the entity-relationship diagram, but that name will be modified in a way that is still auto-explicative once read this design document.

3.2.2 Professor Holds Course relationship

Since we want this relationship to be bidirectional, a new table was added by JEE framework called *Professor_Course* that contains the primary keys of both *Professor* and *Course* table*s*. Each time a new course will be added, the framework will add a new row also in this table.

3.2.3 Project Contains Deliverable relationship

Since we want this relationship to be bidirectional, a new table was added by JEE framework called *Project_Deliverable* that contains the primary keys of both *Project* and *Deliverable* table*s*. Each time a new deliverable will be added, the framework will add a new row also in this table.

3.2.4 Project Contains Deliverable relationship

Since we want this relationship to be bidirectional, a new table was added by JEE framework called *Project_Deliverable* that contains the primary keys of both *Project* and *Deliverable* table*s*. Each time a new deliverable will be added, the framework will add a new row also in this table.

3.2.5 Project Team Uploads Artifact relationship

Since we want this relationship to be bidirectional, a new table was added by JEE framework called *Team_Artifact* that contains the primary keys of both *Team* and *Artifact* table*s*. Each time a new artifact will be added, the framework will add a new row also in this table.

3.2.6 Deliverable Refers To Artifact relationship

Since we want this relationship to be bidirectional, a new table was added by JEE framework called *Deliverable_Artifact* that contains the primary keys of both *Deliverable* and *Artifact* table*s*. Each time a new artifact will be added, the framework will add a new row also in this table.

3.3 Server Module

Identification	Entities
Type	Package
Purpose	Provides the classes that will be mapped into tuples in the corresponding tables of the database.
Function	This is simply a package of classes. Each class is a *JEE Entity*. An entity is a lightweight persistence domain object. Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The persistent state of an entity is represented either through persistent fields or persistent properties. These fields or properties use object/relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.
Subordinates	Inside this package there are the following classes: • **Artifact** • **ArtifactId** • **Course** • **Deliverable** • **DeliverableId** • **Professor** • **Project** • **ProjectId** • **Team** • **TeamId** • **Student** • **User**
Dependencies	The Server software use the class package to know the representation of every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database. In particular, there is a huge amount of data requested and modified by the session beans class in the package *Sessions*.
Interfaces	All classes in the class library will implement the Serializable interface, so that they can easily be marshalled over the network by the server as well as saved to disk on the client. Each Entity will have its set of accessor and mutator methods.
Resources	The specific resources required for the class library are JDBC and Java's serialization mechanism.
Processing	None.
Data	None.

Table 2: Entities Package

Figure 6: Entities Class Diagram

3.3.1 User

Identification	User
Type	Class
Purpose	Represents an User entity and encapsulates all necessary data for storing an user into the database.
Function	A user has the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • String **username**; • String **password**; • String **firstName**; • String **lastName**; • String **email**; • Date **birthday**; • String **telephoneNumber**; All the attributes map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 17.
Subordinates	Defined in the *Function* sub-section.
Dependencies	For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 3: User Class

3.3.2 Student

Identification	Student
Type	Class
Purpose	Represents a Student entity and encapsulates all necessary data for storing an user into the database.
Function	The class *Student* is a subclass of the *User* class, thus it has the *User* data items plus the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • Set<Team> **studentTeams**; • Set<Team> **studentTeamEnrollingRequests**; This two fields represent the *StudentBelongsTeam* and *StudentRequestsMembership* tuple: they are in fact a set of values for each instance of student (whose instance represent a single tuple in the student table), thus realizing a many to many relationship. For more information about this two relationships, see the the Detailed Table Description on page 17.
Subordinates	Defined in the *Function* sub-section.
Dependencies	The Student class has an association with the Team class, because every student can belong to a particular team modeled by the *Team* class as shown in the Server Module package class diagram on page 23. For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 4: Student Class

3.3.3 Professor

Identification	Student
Type	Class
Purpose	Represents a Professor entity and encapsulates all necessary data for storing an user into the database.
Function	The class *Professor* is a subclass of the *User* class, thus it has the *User* data items plus the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • Set<Course> **coursesHeld**; This field represent the one to many relationship between the Course entity and the Professor entity. For more information about this two relationships, see the the Detailed Table Description on page 17.
Subordinates	Defined in the *Function* sub-section.
Dependencies	The Professor class has an association with the Course class, because every course is held by one and only one professor, as shown in the Server Module package class diagram on page 23. For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 5: Professor Class

3.3.4 Course

Identification	Course
Type	Class
Purpose	Represents a Course entity and encapsulates all necessary data for storing an user into the database.
Function	The class *Course* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • protected Long **uid**; • protected String **name**; • protected String **description**; • protected Professor **professor**; All the fields map the corresponding column in the database. In particular, the **professor** field maps the one to many association with the Professor entity. For more information about their meaning, see the the Detailed Table Description on page 17.
Subordinates	Defined in the *Function* sub-section.
Dependencies	The Course class has an association with the Professor class, because every course is held by one and only one professor, as shown in the Server Module package class diagram on page 23. For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 6: Course Class

3.3.5 Project

Identification	Project
Type	Class
Purpose	Represents a Project entity and encapsulates all necessary data for storing an user into the database.
Function	The class *Project* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private ProjectId **id**; • private Date **startDate**; • private Date **endDate**; • private String **description**; All the fields map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 17. In particular, the **id** field maps the primary key of the Project table. For more information about the key, check the ProjectId on the following page
Subordinates	Defined in the *Function* sub-section.
Dependencies	The Project class is related to the Team, Course, Deliverable classes, because every class listed before is linked to a particular project, as shown in the Server Module package class diagram on page 23. For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 7: Project Class

3.3.6 ProjectId

Identification	ProjectId
Type	Class
Purpose	Represents a Project Identifier, an embeddable object representing the primary key of the Project entity.
Function	The class *ProjectId* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private Course **course**; • private String **title**; The **course** field map the corresponding *entity* in the database, while the field **title** represent the name of the Project. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.
Subordinates	Defined in the *Function* sub-section.
Dependencies	The ProjectId class is strictly related to the Project class, because it represent its identifier.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 8: ProjectId Class

3.3.7 Team

Identification	Team
Type	Class
Purpose	Represents a Team entity and encapsulates all necessary data for storing an user into the database.
Function	The class *Team* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private TeamId **id**; • private boolean **closed**; • Set<Student> **studentList**; • Set<Student> **studentEnrollingRequests**; All the fields map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 17. In particular, the **id** field maps the primary key of the Team table. For more information about the key, check the ProjectId on the preceding page
Subordinates	Defined in the *Function* sub-section.
Dependencies	The Team class is related to the Project and Artifact classes, because every class is linked to a particular team, as shown in the Server Module package class diagram on page 23. For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 9: Team Class

3.3.8 TeamId

Identification	TeamId
Type	Class
Purpose	Represents a Team Identifier, an embeddable object representing the primary key of the Team entity.
Function	The class *TeamId* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private Project **project**; • private String **name**; The **project** field map the corresponding *entity* in the database, while the field **name** represent the name of the Team. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.
Subordinates	Defined in the *Function* sub-section.
Dependencies	The TeamId class is strictly related to the Team class, because it represent its identifier.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 10: TeamId Class

3.3.9 Deliverable

Identification	Deliverable
Type	Class
Purpose	Represents a Deliverable entity and encapsulates all necessary data for storing an user into the database.
Function	The class *Deliverable* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private DeliverableId **id**; • private Date **deadline**; • private int **penalty**; • private String **description**; All the fields map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 17. In particular, the **id** field maps the primary key of the Deliverable table. For more information about the key, check the ProjectId on page 30
Subordinates	Defined in the *Function* sub-section.
Dependencies	The Deliverable class is related to the Project and Artifact classes, because every deliverable is linked to a particular project and every artifact is linked to a particular deliverable, as shown in the Server Module package class diagram on page 23. For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 11: Deliverable Class

3.3.10 DeliverableId

Identification	DeliverableId
Type	Class
Purpose	Represents a Deliverable Identifier, an embeddable object representing the primary key of the Deliverable entity.
Function	The class *DeliverableId* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private Project **project**; • private String **name**; The **project** field map the corresponding *entity* in the database, while the field **name** represent the name of the Deliverable. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.
Subordinates	Defined in the *Function* sub-section.
Dependencies	The DeliverableId class is strictly related to the Deliverable class, because it represent its identifier.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 12: DeliverableId Class

3.3.11 Artifact

Identification	Artifact
Type	Class
Purpose	Represents an Artifact entity and encapsulates all necessary data for storing an user into the database.
Function	The class *Artifact* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private ArtifactId **id**; • private Date **submissionDate**; • private int **score**; • private String **fileName**; All the fields map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 17. In particular, the **id** field maps the primary key of the Artifact table. For more information about the key, check the ProjectId on page 30
Subordinates	Defined in the *Function* sub-section.
Dependencies	The Deliverable class is related to the Deliverable and Team classes, because every artifact is linked to a particular deliverable and every artifact is linked to a particular team who created it, as shown in the Server Module package class diagram on page 23. For more information, see the Dependencies row in the Server Module package table on page 23.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 13: Artifact Class

3.3.12 ArtifactId

Identification	ArtifactId
Type	Class
Purpose	Represents an Artifact Identifier, an embeddable object representing the primary key of the Artifact entity.
Function	The class *DeliverableId* contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods. • private Deliverable **deliverable**; • private Team **team**; The **deliverable** and the **team** fields map the corresponding *entity* in the database. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.
Subordinates	Defined in the *Function* sub-section.
Dependencies	The ArtifactId class is strictly related to the ArtifactId class, because it represent its identifier.
Interfaces	Defined in the *Function* sub-section.
Resources	None.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 14: ArtifactId Class

3.3.13 Session Beans

Identification	Sessions
Type	Package
Purpose	Provides the classes that will control and modify the entity beans in the package Server Module on page 23.
Function	This is simply a package of packages. Each package contains a particular type of *JEE Session bean*. A session bean performs operations, such as calculations or database access, for the client, querying for entities and modifying them. The session contained here can query and modify data related to a professor's tasks or student's tasks.
Subordinates	Inside this package there are the following packages: • **Local** • **Remote**
Dependencies	The Server software use the class package to manipulate every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database.
Interfaces	All classes in the class library will implement the Serializable interface, so that they can easily be marshalled over the network by the server as well as saved to disk on the client. Each Entity will have its set of accessor and mutator methods.
Resources	The specific resources required for the class library are JDBC, Java's serialization mechanism, and the RMIIO library to transfer data transfer objects (DTO's).
Processing	None.
Data	None.

Table 15: Entities Package

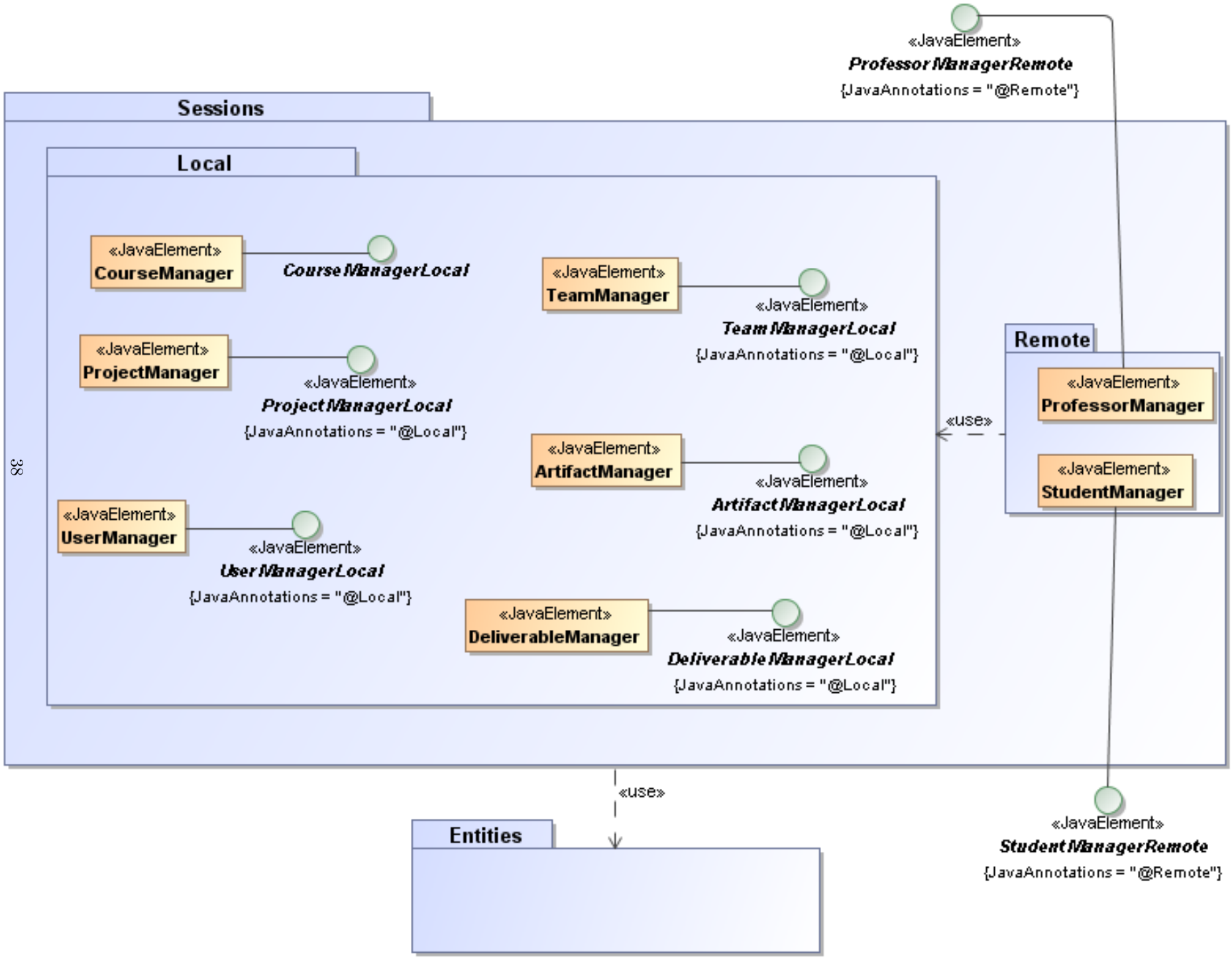

Figure 7: Sessions Class Diagram

3.3.14 Local

Identification	Sessions.Local
Type	Package
Purpose	Provides the classes that will control and modify the entity beans in the package Server Module on page 23.
Function	This is simply a package of classes. Each class is a *local JEE Stateless Session bean.* A local stateless session bean performs operations, such as calculations or database access, and can be accessed **only by the server**. Each session beans will accept commands from one or more remote stateful session bean: each command create or edit entities. In this way, all the stateless session beans of this package implements the JEE design pattern *Data Access Object* (you can find more information about this pattern at thislink).
Subordinates	Inside this package there are the following classes: • **UserManager** • **CourseManager** • **ProjectManager** • **DeliverableManager** • **TeamManager** • **ArtifactManager**
Dependencies	The Server software use the class package to manipulate every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database. These classes will be used by other stateful beans in other to reuse the code between different stateful sessions.
Interfaces	All classes implements their local interfaces, provided in the same package, ad is identified by the name of the class with the suffix "Local". For example, the interface of UserManager is located in the interface UserManagerLocal.
Resources	The specific resources required for the class library are JDBC, Java's serialization mechanism, and the RMIIO library to transfer data transfer objects (DTO's).
Processing	When a stateful session bean from the Remote package sends a command to a stateless session bean from this package, the stateless session bean will check the arguments, then will search for the entities to be modified, then will perform modifications and then will create the return value if requested. For an example you can go and check a sample pseudocode (see A.2) in the appendix of this document.
Data	None.

Table 16: Sessions.Local Package

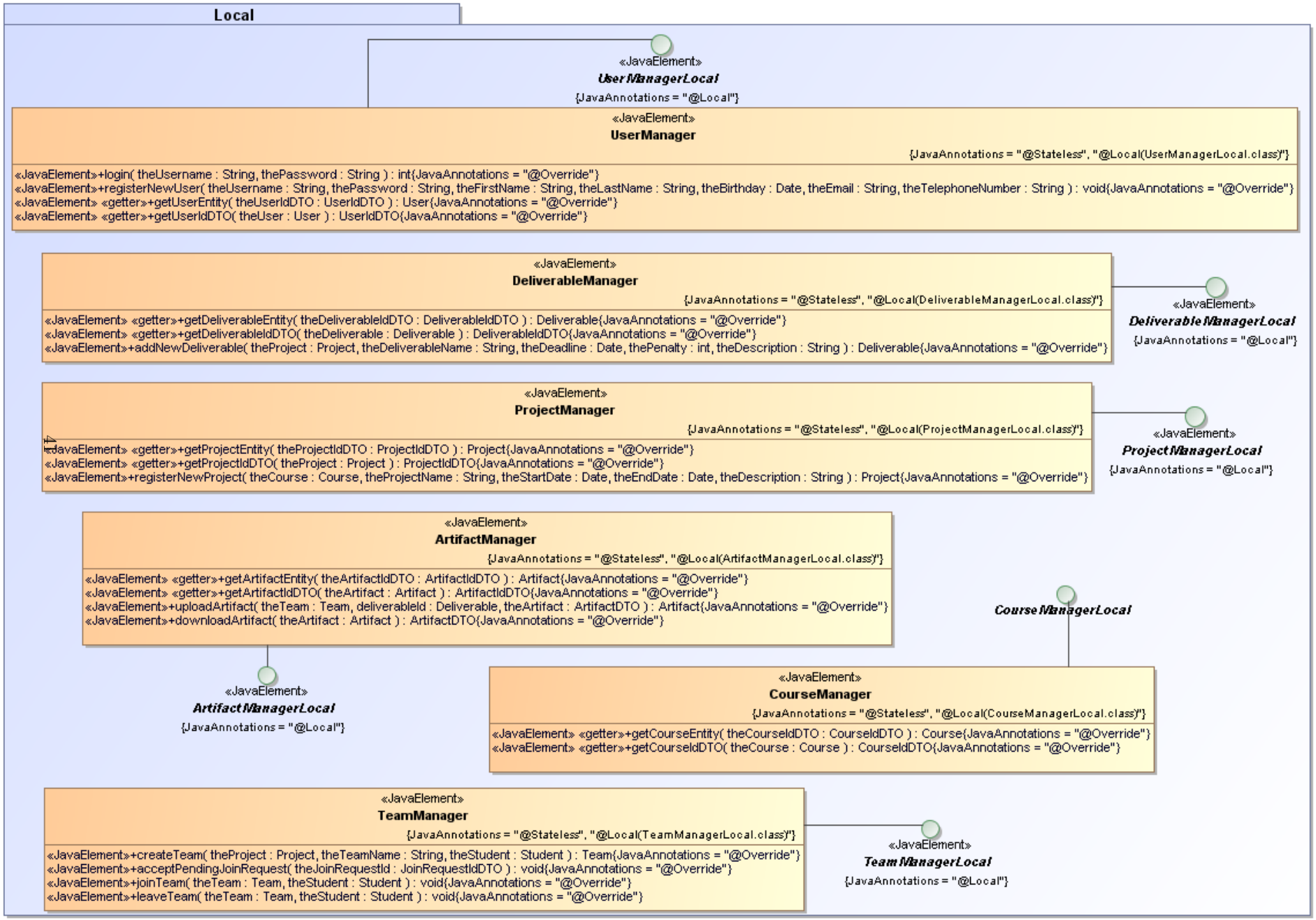

Figure 8: Sessions.Local Class Diagram

3.3.15 Remote

Identification	Sessions.Remote
Type	Package
Purpose	Provides the classes that will execute the client required actions.
Function	This is simply a package of classes. Each class is a *remote JEE Session bean*. A remote session bean performs operations, such as calculations or database access, and can be accessed by the client. Each command sent by the client is translated into several commands executed locally by the local package. In this way, all the session beans of this package implements the design pattern *Facade*.
Subordinates	Inside this package there are the following packages: • **StudentManager** • **ProfessorManager**
Dependencies	The Server software use the class package to manipulate every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database. The classes will provide an implementation of the logic business of the software.
Interfaces	All classes in the package implement a shared interface with the client, contained in the shared class library.
Resources	The specific resources required for the class library are JDBC, Java's serialization mechanism, and the RMIIO library to transfer data transfer objects (DTO's).
Processing	For each command sent by the client, some business logic checks are performed in order to execute the command, then the command is translated into several commands executed locally by the local package. For an example you can go and check a sample pseudocode (see A.3) in the appendix of this document.
Data	None.

Table 17: Sessions.Remote Package

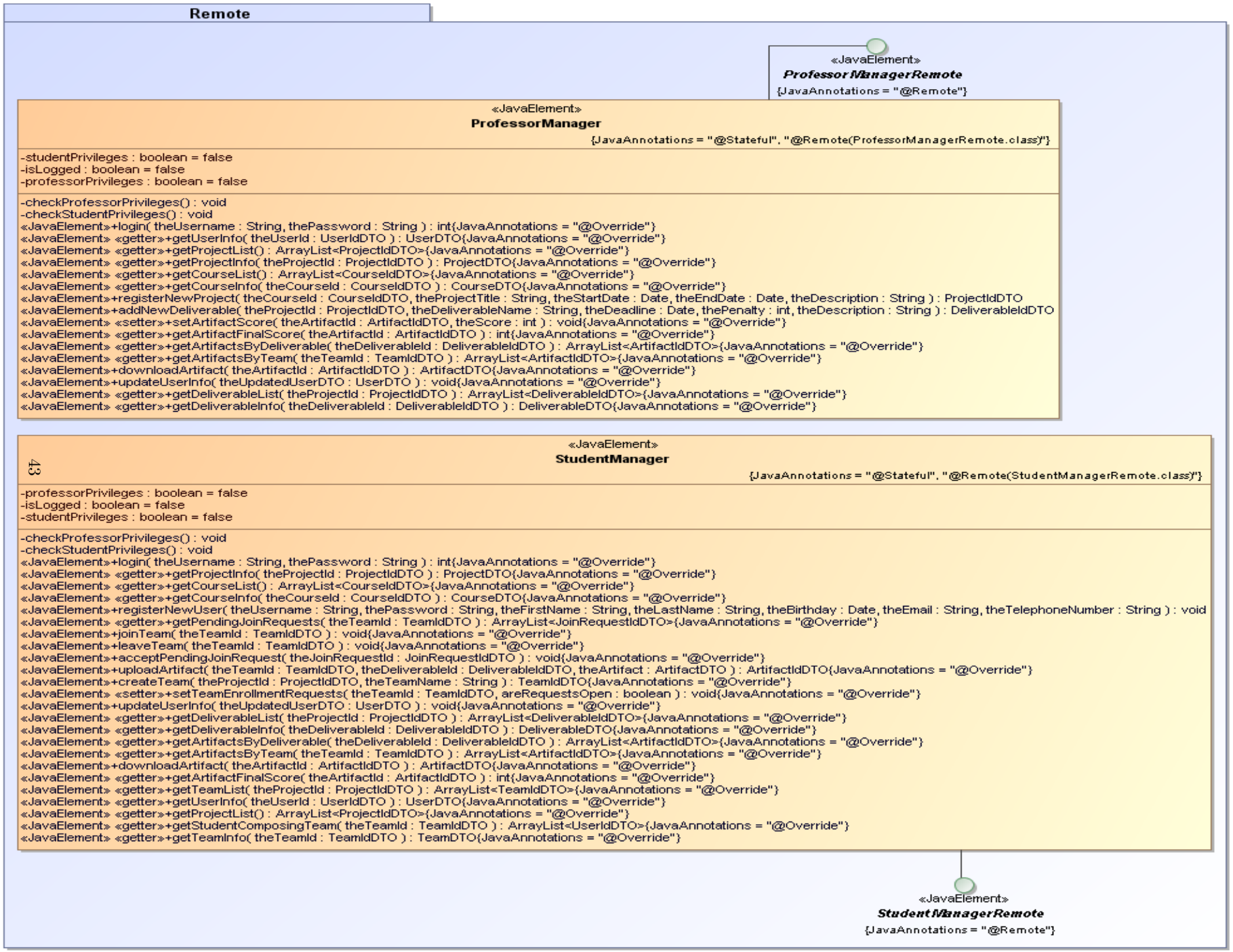

Figure 9: Sessions Remote Class Diagram

3.4 Shared Module

Identification	Shared
Type	Module
Purpose	This module contains classes for the interaction between the Server and Client modules. It provides Data Transfer Objects and a Server Interface for the client, which will use to get receive data from the Server.
Function	• Provide DTO which contains the data transferred during the communication between the Server and Client modules • Contain the exceptions classes used by the system modules • Provide a Server interface for the Client
Subordinates	It contains the following packages: • DTO • Exceptions • Sessions • Util
Dependencies	The Shared Module does not depend on any other system modules, instead the other two modules depend on this one.
Interfaces	• ProfessorManagerRemote.java • StudentManagerRemote.java • UserManagerRemote.java
Resources	The RMIIO library used for file transfer.
Processing	None.
Data	Defined in the *Function* sub-section.

Table 18: Shared Module

3.4.1 Data Transfer Objects

Identification	dto
Type	Package
Purpose	A Data Transfer Object is used to encapsulate the business data. A single method call is used to send and retrieve the Transfer Object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the Transfer Object, populate it with its attribute values, and pass it by value to the client. This objects implements the JEE Pattern **Data Transfer Object**.
Function	• Provide DTO which contains the data transferred during the communication between the Server and Client modules • Contain the exceptions classes used by the system modules • Provide a Server interface for the Client
Subordinates	It contains the dto.ids Package (3.4.2) and the following classes: • **ArtifactDTO** • **CourseDTO** • **DeliverableDTO** • **ProjectDTO** • **TeamDTO** • **UserDTO**
Dependencies	The Client will use DTO ID's to identify a DTO and retrieve information from it, so the class that will implement the ProfessorManagerRemote and StudentManagerRemote interfaces will use these DTO to send and receive data.
Interfaces	They all implement the Serializable interface.
Resources	None.
Processing	None.
Data	DTO are initialized by their constructors, the data are given as parameters.

Table 19: DTO Package

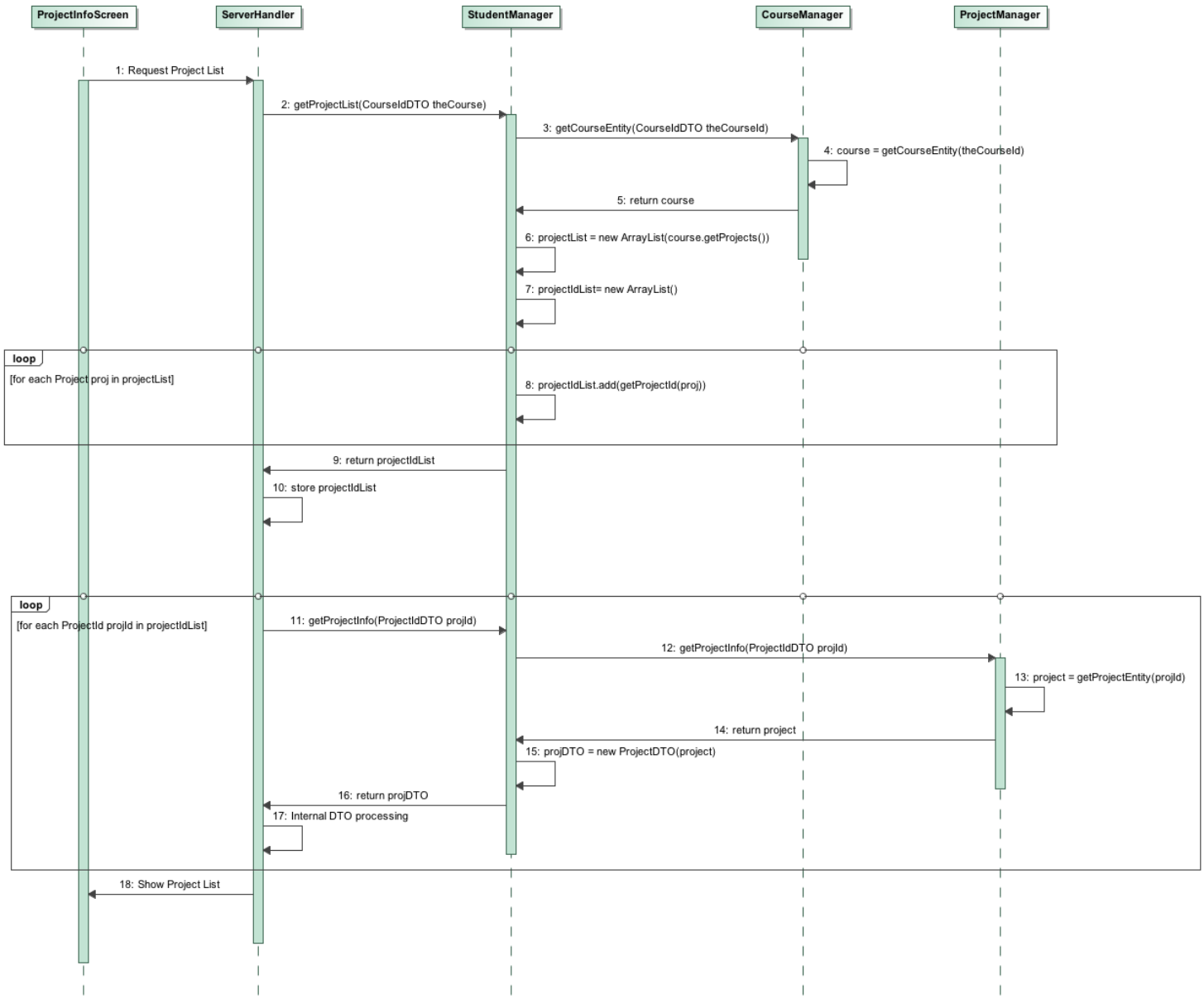

Figure 10: Example of DTO passing between Client and Server

3.4.2 Data Transfer Objects Identifiers

Identification	dto.ids
Type	Package
Purpose	When an enterprise bean uses a Transfer Object, the client makes a single remote method invocation to the enterprise bean to request the Transfer Object instead of numerous remote method calls to get individual attribute values. The enterprise bean then constructs a new Transfer Object instance, copies values into the object and returns it to the client. The client receives the Transfer Object through the DTO ID and can then invoke getter methods on the Transfer Object to get the individual attribute values from the Transfer Object. Because the Transfer Object is passed by value to the client, all calls to the Transfer Object instance are local calls instead of remote method invocations.
Function	This is simply a package of classes. Each class is a *Data Transfer Object ID*.
Subordinates	Inside this package there are the following classes: • **ArtifactIdDTO** • **CourseIdDTO** • **DeliverableIdDTO** • **JoinRequestIdDTO** • **ProfessorIdDTO** • **ProjectIdDTO** • **StudentIdDTO** • **TeamIdDTO** • **UserIdDTO**
Dependencies	The Client will use DTO ID's to identify a DTO and retrieve information from it.
Interfaces	They all implements the *Serializable* interface.
Resources	None.
Processing	None.
Data	DTO are initialized by their constructors, the data are given as parameters.

Table 20: DTO Package

3.4.3 Exceptions

AlreadyLoggedInException occurs when there is an attempt to login using the credentials of a user already logged in

CourseNotHeldException occurs when a professor tries to execute operations for a course which it is not held by him/her

DownloadErrorException occurs when there is an error during the download of an artifact by a Client

InvalidEmailException occurs when the inserted email address does not respect the syntax $local-part@domain$

EntityNotFoundException occurs when the searched entity is not found in the Database

FailedRegistrationException occurs when the registration procedure fails

FilePathNotFoundException occurs when the file path of an artifact is incorrect

ForbiddenOperationException occurs when the entity invoking a method does not have the rights to perform that operation

InconsistentDateException occurs when there is an insertion of date inconsistent with the current date or the ones stored in the Database

NegativeValueException occurs when a negative value is inserted for data which are positive by definition

PasswordMismatchException occurs when the password used to log in does not match with the one stored in the Database

ProjectNotOwnedException occurs when a professor tries to execute operations for a project whose he/she is not the owner

StudentNotEnrolledException occurs when a student tries to execute operations in a project team that he/she does not belong to

TeamClosedException occurs when there is an attempt to join a team which does not accept any new membership requests

UploadErrorException occurs when there is an error during the upload of an artifact to the Server

UsernameAlreadyExistsException occurs when there is an attempt of a student registration with the same username of an already existing user

3.4.4 Sessions

Identification	sessions
Type	Package
Purpose	This package contains the interfaces implemented by the Server session beans. They are grouped by functionality, this ensures the modularization of the system, making easy for a developer to add functionalities by simply adding new interfaces. This decomposition makes sure that the Client does not know the real methods invoked but only the list of functionalities it can access.
Function	• Provide protected remote interface extended by the Professor and Student remote interfaces
Subordinates	It contains the following sets of interfaces: • Artifact 3.4.4.1 • Course 3.4.4.2 • Deliverable 3.4.4.3 • Project 3.4.4.4 • Team 3.4.4.5 • User 3.4.4.6 and the remote interfaces which extend the needed interfaces: • ProfessorManagerRemote.java • StudentManagerRemote.java
Dependencies	The Server session beans implement all the methods declared in the functionalities interfaces.
Interfaces	ProfessorManagerRemote and StudentManagerRemote implement different functionalities interfaces according to the specifications.
Resources	None.
Processing	None.
Data	None.

Table 21: Sessions Package

3.4.4.1 Artifact

- ArtifactEditingFunctionalities.java
- ArtifactEvaluatorFunctionalities.java
- ArtifactViewerFunctionalities.java

3.4.4.2 Course

- CourseViewerFunctionalities.java

3.4.4.3 Deliverable

- DeliverableEditingFunctionalities.java
- DeliverableViewerFunctionalities.java

3.4.4.4 Project

- ProjectEditingFunctionalities.java
- ProjectViewerFunctionalities.java

3.4.4.5 Team

- TeamBasicFunctionalities.java
- TeamCreationFunctionalities.java
- TeamManagementFunctionalities.java
- TeamViewerFunctionalities.java

3.4.4.6 User

- UserEditingFunctionalities.java
- UserLoginFunctionalities.java
- UserRegisteringFunctionalities.java
- UserViewerFunctionalities.java

3.5 Client Module

Identification	Client
Type	Module
Purpose	The Client is the application used by the users (students and professor) to connect to the MPH system and manage their project stuff. It provides a GUI and a RMI connection manager to send commands to the Server.
Function	• Let a professor publish a new project and evaluate the artifacts delivered by the corresponding project teams • Let a student register and log into the MPH system • Let a student manage his/her belonging project team and upload artifacts corresponding to the project deliverables • Send commands to the Server to get and save the required data
Subordinates	Connection Package 3.5.1 Student GUI Package 3.5.5 Professor GUI Package 3.5.6
Dependencies	This component will interact closely with the Server Module for the retrieval of data. Server calls will be synchronous, so there will be no timing issues for the Client to consider. The Client will use various data classes and interfaces found in the Shared Class Library. Data from the Server will be packaged into one of these classes, and the Client can then access that data through the public methods of an appropriate class. See 3.4.1.
Interfaces	The Client uses Swing GUI classes for interaction with the user. The screen formats for these windows are shown in the User Experience Report. It uses the ProfessorManagerRemote and StudentManagerRemote interfaces for interaction with the Server. See 3.4.4.
Resources	It will contain the Shared Module in the build path as it will invoke some of the methods contained in the ProfessorManagerRemote and StudentManagerRemote. See 3.4.4.
Processing	None.
Data	Data received from the user are stored locally then sent via RMI to the Server, which will save them permanently in the Database. Viceversa data requested by the user are sent via RMI by the Server Module.

Table 22: Client Module

3.5.1 Connection Package

Identification	Connection
Type	Package
Purpose	This package contains all the logic of the Client.
Function	• Provide methods to establish a connection between a Client and the MPH Server • Provide methods to get and save data from and to the MPH Server
Subordinates	• Connection Manager • ProfessorManager • StudentManager
Dependencies	Client applications will use this package to retrieve and synchronize data with the server. RMI will be used for the communication thus this package has to invoke methods present in the Shared Class Library.
Interfaces	ProfessorManager and StudentManager implement the respective remote interfaces present in the Shared module. See 3.4.4
Resources	The communication will use the TCP/IP network protocol, relying on Java's RMI. It also relies on Java's ObjectStream class and Java's serialization mechanism.
Processing	None.
Data	The DTO's containing the data to send or to receive will be serialized.

Table 23: Connection Package

3.5.2 ConnectionManager

Identification	ConnectionManager
Type	Class
Purpose	This package manages the communication between Client and Server applications. It is responsible to establish a RMI connection with the MPH Server.
Function	• Provide methods to establish a connection between a Client and the MPH Server
Subordinates	• The initial context for the RMI connection with the MPH Server
Dependencies	StudentManager and ProfessorManager will use this class to set up a connection with the server.
Interfaces	None.
Resources	The communication will use the TCP/IP network protocol, relying on Java's RMI. It also relies on Java's ObjectStream class and Java's serialization mechanism.
Processing	None.
Data	None.

Table 24: ConnectionManager

3.5.3 ProfessorManager

Identification	ProfessorManager
Type	Class
Purpose	This class contains the methods invoked by the Professor GUI to retrieve information and save data to the MPH Server.
Function	• Provide methods to send commands to the server to the GUI
Subordinates	• The methods contained in the ProfessorManagerRemote interface.
Dependencies	It uses ConnectionManager to set up a connection with the server.
Interfaces	It implements the ProfessorManagerRemote interface.
Resources	The communication will use the TCP/IP network protocol, relying on Java's RMI. It also relies on Java's ObjectStream class and Java's serialization mechanism.
Processing	None.
Data	The DTO's containing the data to send or to receive will be serialized.

Table 25: ProfessorManager

3.5.4 StudentManager

Identification	StudentManager
Type	Class
Purpose	This class contains the methods invoked by the Student GUI to retrieve information and save data to the MPH Server.
Function	• Provide methods to send commands to the server to the GUI
Subordinates	• The methods contained in the StudentManagerRemote interface.
Dependencies	It uses ConnectionManager to set up a connection with the server.
Interfaces	It implements the StudentManagerRemote interface.
Resources	The communication will use the TCP/IP network protocol, relying on Java's RMI. It also relies on Java's ObjectStream class and Java's serialization mechanism.
Processing	None.
Data	The DTO's containing the data to send or to receive will be serialized.

Table 26: StudentManager

3.5.5 Student GUI Package

Identification	GUI.Student
Type	Package
Purpose	This package contains all the classes used for the GUI
Function	It provides a graphical interface to: • let the professor publish a new project • let the professor evaluate the artifacts delivered by the project teams
Subordinates	RegistrationScreen 3.5.7 LoginScreen 3.5.8 StudentMainScreen 3.5.9 StudentInfoScreen 3.5.10 ProjectSelectionScreen 3.5.11 ProjectInfoScreen 3.5.12
Dependencies	The GUI will use the Client Module to get and put data on the server.
Interfaces	The GUI uses Swing classes for interaction with the user. The screen formats for these windows are shown in the User Experience Report.
Resources	The GUI will not have any significantly CPU or memory-intensive tasks however, normal overhead associated with Swing components is to be expected in both cases. Much of the project data to be queried will be accessed on a remote database.
Processing	See the following UX diagram.
Data	The data inserted by the users are sent to the Server via RMI which will store them permanently in the Database.

Table 27: GUI.Student Package

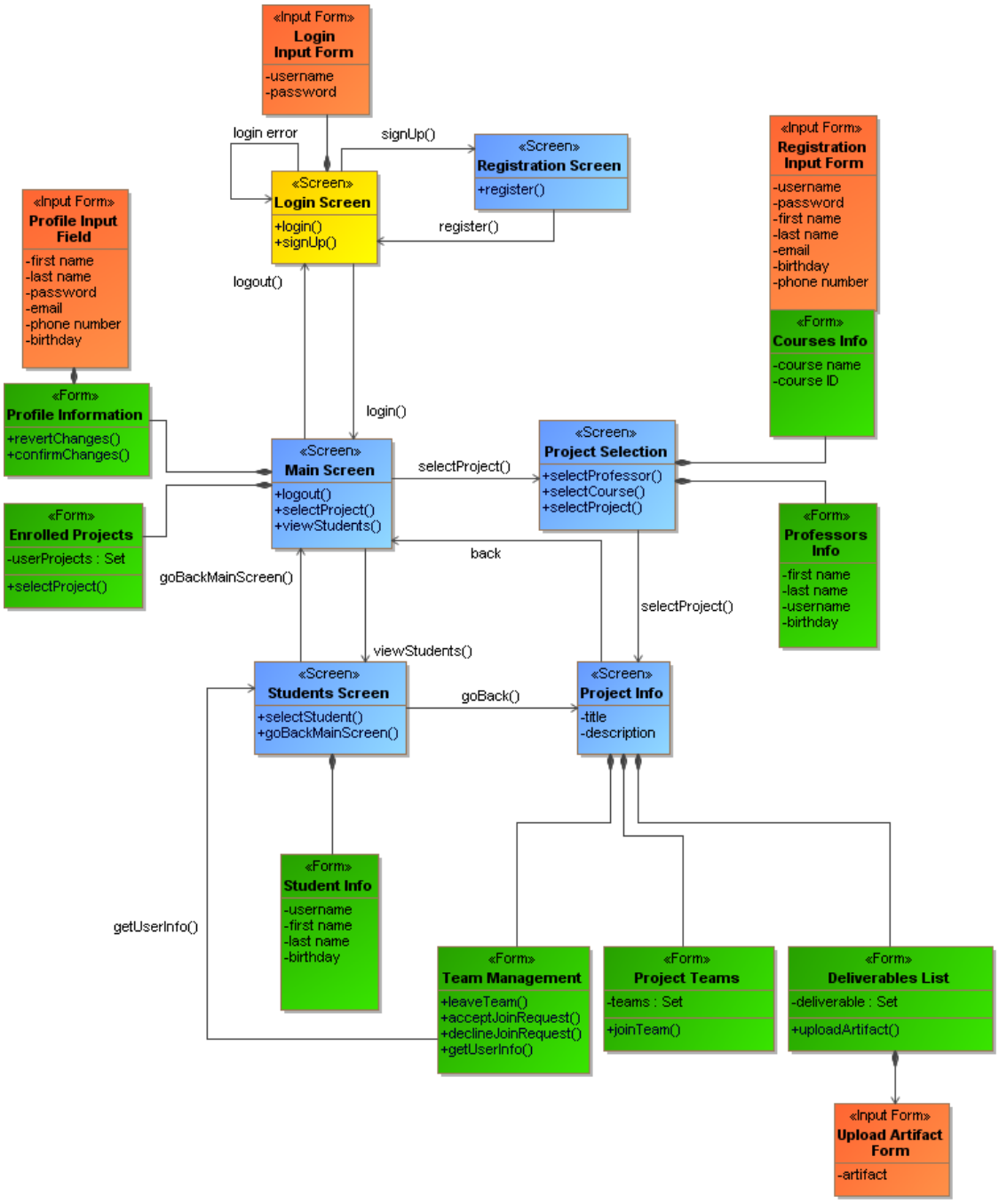

Figure 11: Student UX Diagram

3.5.6 Professor GUI Package

Identification	GUI.Professor
Type	Package
Purpose	This package contains all the classes used for the GUI
Function	It provides a graphical interface to: • let the student join project teams and manage them • let the student upload a new artifact
Subordinates	LoginScreen 3.5.8 ProfessorMainScreen 3.5.13 NewProjectScreen 3.5.14 ArtifactsScreen 3.5.15 ProjectInfoScreen 3.5.12
Dependencies	The GUI will use the Client Module to get and put data on the server.
Interfaces	The GUI uses Swing classes for interaction with the user. The screen formats for these windows are shown in the User Experience Report.
Resources	The GUI will not have any significantly CPU or memory-intensive tasks however, normal overhead associated with Swing components is to be expected in both cases. Much of the project data to be queried will be accessed on a remote database.
Processing	See the following UX diagram.
Data	The data inserted by the users are sent to the Server via RMI which will store them permanently in the Database.

Table 28: GUI.Professor Package

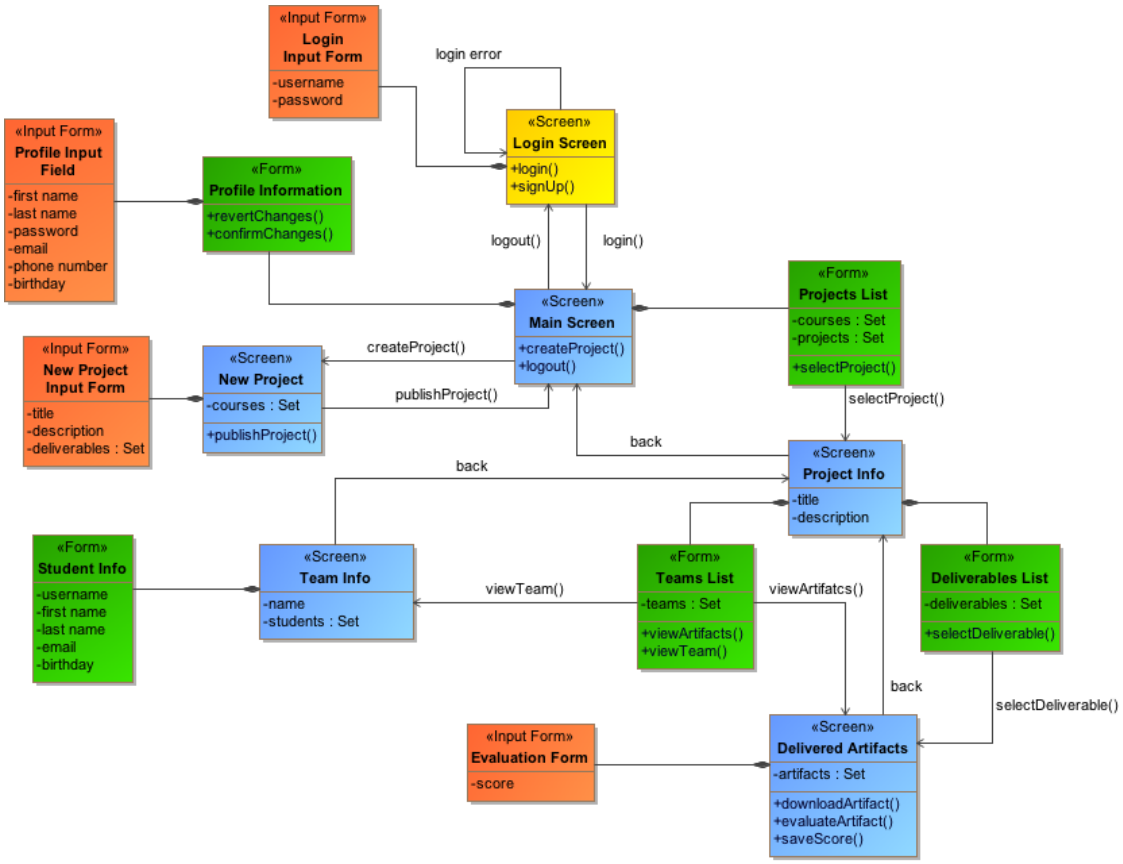

Figure 12: Professor UX Diagram

3.5.7 RegistrationScreen

Identification	Registration Screen
Type	Class
Purpose	This lets the Student register to the MPH system
Function	• Register a new student into the MPH system • Send to the server application the student registration data via RMI
Subordinates	It contains • a username text field • two password fields for security reasons • optional text fields for personal information • a "Register" button which takes the student to the Login Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See User Experience Report.
Data	The student username, password and personal data are serialized and sent to the server which will store them in the Database.

Table 29: Registration Screen

3.5.8 LoginScreen

Identification	LoginScreen
Type	Class
Purpose	This lets a registered Student log into the MPH system
Function	• Log in a student into the MPH system • Send to the server application the student login credentials via RMI
Subordinates	It contains • username and password text fields editable by the user • a "Log In" button which, if authentication succeeds, takes the student to the Main Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See UX.html.
Data	The student username and password are serialized and sent to the server which will check them against the credentials saved in the Database.

Table 30: Login Screen

3.5.9 StudentMainScreen

Identification	StudentMainScreen
Type	Class
Purpose	This shows the student the MPH resources he/she can access
Function	• Let a student edit his/her personal data • Let a student manage the project teams which he/she belongs to • Let the student search for information about other students, courses and professors
Subordinates	It contains • a "Profile Information" form with personal information and buttons to edit them • a "Select Project" button which takes the student to the Project Selection Screen • a "Enrolled Projects" form containing a list of clickable projects which take the student to the Project Info Screen • a "Search for Students" button which takes the student to the Students Screen • a "Log Out" button which takes the student to the Login Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See User Experience Report.
Data	No data are exchanged in this class.

Table 31: Student Main Screen

3.5.10 StudentInfoScreen

Identification	StudentInfo Screen
Type	Class
Purpose	This shows the student personal information about other students registered in the MPH system
Function	• Let a student view other student's personal data
Subordinates	It contains • a list of students registered into the MPH system • the student names are clickable and clicking shows the information about the selected student • a "Back" button which takes the student to the Main Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See User Experience Report.
Data	The search criteria are sent to the Server which will send the corresponding results taken from the Database

Table 32: Students Screen

3.5.11 ProjectSelectionScreen

Identification	ProjectSelectionScreen
Type	Class
Purpose	This lets the student view a list of projects referring to a selected course held by a professor
Function	• Let a student view a list of projects
Subordinates	It contains • a list of selectable professors; clicking on one of them opens a form containing the professor information • a list of selectable courses held by the selected professor; clicking on one of them opens a for containing the professor information • a list of clickable projects referring to the selected projects which take the student to the Project Info Screen • a "Back" button which takes the student to the Main Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See User Experience Report.
Data	The search criteria are sent to the Server which will send the corresponding results taken from the Database.

Table 33: Project Selection Screen

3.5.12 ProjectInfoScreen

Identification	ProjectInfoScreen
Type	Class
Purpose	This shows the student the project information, allowing him/her to view the project deliverables, manage the project team he/she belongs to or join a new project team.
Function	• Let a student view the project information and deliverables • Let a student see the list of teams referring to a project and join one of them • Let a student manage his/her project team
Subordinates	This contains • the description of the previously selected project • a "Deliverables List" form containing – a list of deliverables – a "Upload Artifact" button next to each deliverable which lets the student upload a file from his/her computer • a "Team Management" form containing – a list of incoming membership requests by other students – a "Accept Join Request" next to an incoming membership request – a "Decline Join Request" next to an incoming membership request – a "Leave Team" button which lets the student leave the project team • a "Project Teams" form containing – a list of teams enrolled in the previously selected project – a "Join Team" button which sends a new Membership Request to the team members • a "Back" button which takes the student to the Main Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See User Experience Report.
Data	The project data are sent by the server which took them from the Database.

Table 34: Project Info Screen

3.5.13 ProfessorMainScreen

Identification	ProfessorMainScreen
Type	Class
Purpose	This shows the professor the MPH resources he/she can access
Function	• Let a professor see the list of courses held by him/her • Let the professor publish a new project
Subordinates	It contains • a frame containing the professor personal information which he/she can edit • a "Publish New Project" button which takes the professor to the New Project Screen • a list of courses and the corresponding projects published by him/her. Selecting a project takes the professor to the Project Info Screen • a "Log Out" button which takes the professor to the Login Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See User Experience Report.
Data	No data are exchanged in this class.

Table 35: Professor Main Screen

3.5.14 NewProjectScreen

Identification	NewProjectScreen
Type	Class
Purpose	This allows the professor to publish a new project for the desired course so the students can create teams to develop the project.
Function	• Let a professor publish a new project for a course • Let the professor define the title and deliverables for the new project
Subordinates	It contains • a list of selectable courses held by the professor • text fields for the project title, deliverables' deadlines' dates and description • a "Publish" button which takes the professor to the Main Screen • a "Back" button which takes the professor to the Main Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See UX.html.
Data	The new project details are sent to the Server via RMI, which will store them permanently in the Database.

Table 36: New Project Screen

3.5.15 ArtifactsScreen

Identification	ArtifactsScreen
Type	Class
Purpose	This allows the professor to view the delivered files per deliverable type or per teams and to evaluate them.
Function	• Let the professor download the artifacts delivered by the project teams • Let the professor evaluate the delivered artifacts
Subordinates	It contains • a list of delivered artifacts corresponding to the selected deliverable or artifacts delivered by the selected team • a "Evaluate" button next to each delivered artifact which opens a small text field where the professor can evaluate the artifact with a score from 1 to 10 • a "Save" button appear after the "Evaluate" button is pressed allowing the professor to save the assigned score • a "Download" button next to each delivered artifact which lets the professor download the artifact • a "Back" button which takes the professor to the Main Screen
Dependencies	Same as other GUI interfaces.
Interfaces	Same as other GUI interfaces.
Resources	Same as other GUI interfaces.
Processing	See User Experience Report.
Data	The delivered artifacts can be downloaded via RMI. The scores assigned by the professor are sent to the server which will store them permanently.

Table 37: Artifacts Screen

3.5.16 Support classes contained in the GUI package

- ComparableDeliverable

  - This class allows the sorting of a set of deliverables. It is used by the ProjectInfoScreen in the Student GUI to view the list of project deliverables sorted by deadline date

- FolderNode
  - This class is needed because Swing JTree assigns a "leaf" icon to every node which hasn't got any child but the development team wanted to display a "folder" icon to all Professor and Course nodes even the ones without children nodes

- TreeEntity
  - This class displays a "nice" name for the objects contained in the Jtree by including the DTO and overriding the toString() method.

4 Reuse and relationship to other products

4.1 Java and Java tools

4.1.1 Programming Language

The customer forced the use of the Java programming language and specifically the JEE architecture.

4.1.2 External Libraries

The developer team set the goal of making use of any existing code or COTS to avoid wasting time reinventing the wheel. The team decided to use open source or freeware solutions wherever possible. Because the team is creating software for a defined application, it is possible to use open source technologies in each area.

4.1.2.1 RMIIO The developer team decided to to make use of the **RMIIO** library in order to exchange files between the Server and Client modules.

RMIIO is an open source library which makes it as simple as possible to stream large amounts of data using the RMI framework. The RMI framework makes it very easy to implement remote communication between Java programs but it lacks an immediate solution for the exchange of files over the network.

What is really needed to do is stream data from the Client to the Server, using a framework which does not really expose a streaming model. The RMIIO library was written to fill in that missing gap in the RMI framework. It provides some very powerful classes which enable a Client to stream data to the Server using only a few extra lines of code.

4.1.2.2 JUnit

The developer team decided to to make use of the **JUnit** library in order to test the project.

JUnit is a unit testing framework for the Java programming language. Even if JUnit is not recommended to test JEE Projects, is easy to use and flexible enough to let a comprehensive test of the project.

4.1.3 IDE

The developer team chose a development environment that would allow to edit the Java code. **Eclipse** was chosen because it is free and open source and has many plug-ins which allow the use of already existing applications.

The team chose the free Eclipse plug-in **WindowBuilderPro** to create the GUI because it allows to drag and drop frames and edit the generated code all within the same environment.

4.2 Database and Reporting

The project required a Database to store the records. The developer team chose MySQL, because it is open source, has many management tools, and includes **MySQL Workbench**, a DBMS management tool that provides a visual console to easily administer MySQL environments and gain better visibility into databases in a easy but still powerful way.

5 Design Decisions and Tradeoffs

5.1 Three Tier Design

Deciding how to judiciously divide the project between all team members was another design issue. We finally decided on a 3 tier design, which is an application program organized into three major parts: the Database Management System (Data Tier), the Server Module (Business Tier), and the Client Module (Presentation Tier).

A 3-tier application uses the client/server computing model. Since the Database is "virtually" included in the Server module the Client and Server can be developed concurrently by a different programmer. Because the programming for a tier can be changed or relocated without affecting the other tiers, the 3-tier model makes it easier for an enterprise or software packager to continually evolve an application as new needs and opportunities arise. Existing applications or critical parts can be permanently or temporarily retained and encapsulated within the new tier of which it becomes a component. This design idea was very appealing to our team, especially for portability purposes.

A Pseudocode and Code Examples

A.1 DBMS Processing Example

```
1  public TeamIdDTO createTeam(ProjectIdDTO theProjectId, String theTeamName)
2                          throws IllegalArgumentException,
                                ForbiddenOperationException {
3
4  //*** Preliminary operations... ***
5
6  //*** Querying the database ***
7          Query q = manager.createQuery("FROM Project p WHERE p.id.course.uid =
                 :n AND p.id.title = :m");
8
9  //*** Set query parameters ***
10         q.setParameter("n", aCourseId);
11         q.setParameter("m", aProjectTitle);
12
13         try {
14                 //Execute query and check the results
15                 if (q.getResultList().isEmpty() == false) {
16
17                         Project aProject = (Project) q.getSingleResult();
18                         Team newTeam = new Team(aProject, theTeamName);
19
20                         //Modify the database state
21                         manager.persist(newTeam);
22                         manager.flush();
23
24                 }
25         }
26         catch (EntityNotFoundException exc) { //Some operations...}
27         catch (NonUniqueResultException exc) { //Some operations... }
28
29 //*** Aftermath operations... ***
30
31 }
```

A.2 Local Session Bean Processing Example

```
1  public ArtifactIdDTO uploadArtifact(TeamIdDTO theTeamIdDTO, DeliverableIdDTO
       theDeliverableIdDTO, OutcomingFileDTO theOutcomingFileDTO, StudentIdDTO
       theStudentIdDTO)
2                  throws InvalidArgumentException, ForbiddenOperationException,
                        UploadErrorException {
3
4  //*** Validating all the argument passed ***
5
6          System.out.println("Trying to upload a new artifact: " +
               theOutcomingFileDTO);
7          System.out.println("Perform some checks");
8          ObjectUtility.isNull(theTeamId);
9          ObjectUtility.isNull(theDeliverableId);
10         ObjectUtility.isNull(theOutcomingFileDTO);
11
12         System.out.println("Trying to validate team");
13         Team aTeam = teamManager.reconstructEntity(theTeamId);
14         ObjectUtility.isNull(aTeam);
15         System.out.println("Trying to validate deliverable");
16         Deliverable aDeliverable = deliverableManager.reconstructEntity(
               theDeliverableId);
17         ObjectUtility.isNull(aDeliverable);
18
19
20 //*** Retriving entities or objects needed to perform the command requested
       ***
21
22         System.out.println("Trying to read from the input stream");
23         InputStream aObjectInputStream;
24
25         try {
26                 System.out.println("Trying to wrap the input stream");
27                 aObjectInputStream = RemoteInputStreamClient.wrap(
                       theOutcomingFileDTO.exportInUpload());
28         }
29         catch (IOException e1) {
30                 throw new UploadErrorException("There was an error trying to
                       upload the file. The file was not uploaded to the server
                       .");
31         }
32
33         ByteArrayOutputStream aStream = new ByteArrayOutputStream();
34         System.out.println("Now reading the the input stream");
35
36         int c;
37         try {
38                 while ((c = aObjectInputStream.read()) != -1) {
39                         aStream.write(c);
40                 }
41         } catch (IOException e) {
42                 throw new UploadErrorException("There was an error trying to
                       read the uploaded file. The file was not uploaded to the
                        server.");
43         }
44         finally {
45                 try {
46                         aStream.close();
47                 }
48                 catch (IOException e) {
49                         throw new UploadErrorException("There was an error
                               finalizing the uploaded file. The file was not
                               uploaded to the server.");
50                 }
51         }
52
53         System.out.println("Transformed into a byte array");
```

```
54          byte[] artifactFile = aStream.toByteArray();
55          Artifact anArtifact = null;
56          boolean merge = false;
57          System.out.println("BEFORE the " + aDeliverable.getId().getName() +
                " had " +aDeliverable.getArtifactSet().size() + " uploaded");
58
59  //*** Performing modifications to entities ***
60
61          System.out.println("Lookup for old artifact linked to this
                deliverable");
62
63          if (existsEntity(new ArtifactId(aDeliverable,aTeam))) {
64                  System.out.println("There is already an artifact for this
                        deliverable. Got the old artifact");
65                  anArtifact = getEntity(new ArtifactId(aDeliverable,aTeam));
66                  merge = true;
67          } else {
68                  System.out.println("There are no artifact for this
                        deliverable. Created a new artifact");
69                  System.out.println("Update the blob");
70                  anArtifact = new Artifact(aDeliverable, aTeam, entityUtility.
                        getServerTimestamp(), artifactFile, theOutcomingFileDTO.
                        getFilePath().getName());                   aDeliverable.
                        addArtifact(anArtifact);
71                  aTeam.addArtifact(anArtifact);
72                  }
73
74  //*** Persisting or merging entities ***
75
76          if (merge) {
77                  System.out.println("Update the blob");
78                  anArtifact.setFile(artifactFile);
79                  anArtifact.setFileName(theOutcomingFileDTO.getFilePath().
                        getName());
80                  anArtifact.setSubmissionDate(entityUtility.getServerTimestamp
                        ());
81                  em.merge(anArtifact);
82          } else {
83                  em.persist(anArtifact);
84          }
85
86          System.out.println("Now the deliverable " + aDeliverable.getId().
                getName() +  " has " +aDeliverable.getArtifactSet().size() + "
                uploaded");
87          em.merge(aDeliverable);
88          System.out.println("Flushing");
89          em.flush();
90
91  //*** Creating the requested return value ***
92
93          return anArtifact.getId().getDTO();
94  }
```

A.3 Remote Session Bean Processing Example

```
1   public ArtifactIdDTO uploadArtifact(TeamIdDTO theTeamId, DeliverableIdDTO
        theDeliverableId, OutcomingFileDTO theFileDTO)
2           throws  InvalidArgumentException, ForbiddenOperationException,
3           UploadErrorException, UserNotLoggedInException {
4
5   //*** Performing business logic checks
6
7           checkLoggedInStatus();
8
9   //*** Executing a local stateless session bean command and return the results
10
11          return artifactManager.uploadArtifact(theTeamId, theDeliverableId,
```

```
                theFileDTO, this.<StudentIdDTO>getUserIdDTO());
12
13  }
```