Apply Object-Orientation and UML to the Development of Web-based Learning System

Choe Sun Yong and Zhiming Liu

March 2003
UNU/IIST and UNU/IIST Reports

UNU/IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macau, and was founded in 1991. It started operations in July 1992. UNU/IIST is jointly funded by the Governor of Macau and the governments of the People’s Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macau authorities also supply UNU/IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU/IIST is to assist developing countries in the application and development of software technology.

UNU/IIST contributes through its programmatic activities:

1. **Advanced development projects**, in which software techniques supported by tools are applied,
2. **Research projects**, in which new techniques for software development are investigated,
3. **Curriculum development projects**, in which courses of software technology for universities in developing countries are developed,
4. **University development projects**, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. **Schools and Courses**, which typically teach advanced software development techniques,
6. **Events**, in which conferences and workshops are organised or supported by UNU/IIST, and
7. **Dissemination**, in which UNU/IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU/IIST is on formal methods for software development. UNU/IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU/IIST produces a report series. Reports are either Research [R], Technical [T], Compendia [C] or Administrative [A]. They are records of UNU/IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU/IIST at P.O. Box 3058, Macau or visit UNU/IIST’s home page: [http://www.iist.unu.edu](http://www.iist.unu.edu), if you would like to know more about UNU/IIST and its report series.

Chris George, Acting Director
Apply Object-Orientation and UML to the Development of Web-based Learning System

Choe Sun Yong and Zhiming Liu

Abstract

With the development of Internet technology, the study of theories and methods in the development of distance learning system is being increased. A Web-based learning system can be considered as a client-server system which gives and receives education between the teachers and the students through the Internet and WWW. The Unified Modeling Language (UML) is now widely used for modeling a software at different stages during its development: requirement analysis, design and implementation. This report shows how object-oriented techniques and UML can be used in modelling and development of a web-based learning system.

Keywords: Web-based Learning System, Object-Orientation, UML.
Choe Sun Yong was a Fellow of UNU/IIST (September 2002 - March 2003), on leave of absence from CIAST of DPR of Korea. She has a MSc. in mathematics from Kim Il Sung University. Her current research interests include the software design using Object-Oriented development technology and UML.

Zhiming Liu is a research fellow at UNU/IIST, on leave from the Department of Mathematics and Computer Science at the University of Leicester, England, where he is a lecturer in computer science. His research interests include formal theories and methods of computing systems in general and formal semantics, specification and verification of fault-tolerant, real-time and concurrent systems; and formal techniques for object-oriented system development. E-mail: Z. Liu@iist.unu.edu.
Contents

1 Introduction 1

2 Introduction to eLearning System 2

3 An Overview of OO Software Development with UML 4
   3.1 Requirement Specification 5
   3.2 Creation of Use Case Model 5
   3.3 Creation of a Conceptual Model 5
   3.4 Creation of Interaction Diagram between Objects 6
   3.5 Mapping a Design Class Diagram to Code 7

4 A WbLS system 8
   4.1 A Requirement Analysis of WbLS 8
   4.2 Use Cases and Use Case Diagrams 12
   4.3 Conceptual Model 20
   4.4 A Behavior of WbLS System 27
   4.5 A WbLS Design 44
      4.5.1 Collaboration Diagrams 44
      4.5.2 Design Class Diagrams 59
   4.6 Implementation 64
      4.6.1 Mapping a Design to Code 64
      4.6.2 WbLS Conceptual Model Packages 65

5 Conclusions 71

6 References 71
1 Introduction

The rapid development of the Internet technologies, especially the evolution of World Wide Web has led to tremendous growth in opportunities to teach and learn outside of the traditional classroom-based education. Over the past decade, the Internet has brought Web-based learning and offers many advantages: it is convenient, available at any time of the day, and can be accessed nearly anywhere in the world. Clearly, Web-based or distributed learning, offers a great potential to increase the availability and convenience of education.

The overall purpose of all kinds of distance education is to give and receive education and to establish some kind of two-way communication between the teachers and the students. Through the Internet a distance education can connect students with teachers in different places at any time. A Web-based Learning System for Computer Science subject (WbLS) is an example of distance learning to give students knowledge on computer science subject through web browser; it can be typically used in the universities and for distance students who require knowledge on the computer science. It includes hardware components such as servers (e.g. web server and database server) and a client supporting web browser, software to run the system. Our final purpose is to create the software to run a Web-based Learning System for Computer Science subject.

The problem domain of distance education system is quite complicated and involves many concepts such as course instructors, students, course materials, learning activities, etc. These concepts are also closely related to determine the structure of the system. Such a system needs to be continuously evolved and maintained as teaching materials and strategies need to be updated timely. All these features indicate that the Object-Oriented Rational Use-Case Driven, Incremental and Iterative Development Process (RDP) [JBR00] will be used.

RDP uses UML as its modelling language that has been accepted as a standard by OMG and is now widely used in OO software development. UML can be used for specifying, visualizing, constructing, and documenting the artifacts of a software development. One of the main advantages of UML is that different modelling notations are used at different stages to represent the system at a proper level of abstraction and to describe both of static structure and dynamic behavior of the system. In this report, we consider how object-oriented techniques and UML through the requirement capture and analysis, design, and implementation can be used in the development of a web-based leaning system. As the starting cycle in the RDP process of whole system (in fact it may never be a whole system as it will evolve anyway), we carry out the requirement analysis and design of the sub-system that is only concerned with web-based learning in computer science as a case study.
2 Introduction to eLearning System

In general, Learning Technology Systems (LTS) are learning, education and training systems that are supported by the Information Technology [IEEE/LTSC01a]. Examples of such systems are of Computer Managed Instruction (CMI) Systems, Computer-based Training (CBT) Systems, Intelligent Tutoring Systems and Web-based Instructional Systems. A special kind of LTSs are the Web-based Instructional Systems (WbIS) which are based on the Internet and WWW technologies in order to provide teaching and learning following the open and distance learning paradigm. The purpose of WbIS is to support and partially automate the learning process on a subject field, which might concern, for example, a course, a seminar or even a series of lectures. Users of WbIS differ in their goals, professional background, interests, and knowledge of the subject matter. To optimize the efficiency and effectiveness of the learning process educational systems should adapt to these user characteristics.

The main advantages of WbIS over the traditional education can be stated as:

- No timetable constraints. The students can take the courses whenever they want.
- No classroom constraints. The students are not constrained to be all present at the same time into a classroom;
- A student can progress over the teaching materials at his / her own speed;
- One can use the principle ”develop once - use anywhere”.
- No limitation for the number of students enrolled to take a certain course;
- No geographical limitation of taking courses;
- Web based education is an effective and low cost solution;

The Following figures show components and the overall architecture of Web-base Learning System.
A Web-based adaptive Learning System should have the following functions [Bru98]:

- **Curriculum sequencing**: this is to provide the students with the most suitable individualized sequence of knowledge units to learn, with a sequence of learning tasks (e.g. examples, questions, problems, etc.) to work through. Curriculum sequencing helps the student to find an "optimal path" through the learning material.

- **Adaptive presentation**: to present the individualized learning materials to students by constructing web pages dynamically.
An Overview of OO Software Development with UML

- Adaptive navigation support: this guides students implicitly and gives them the choice of the next knowledge item to be learned and next problem to be solved.

- Interactive problem solving support: this aims at providing the student with intelligent help at each step of problem solving by giving a hint to execute the next step. Such a system should be able to monitor the actions of the student, understand them, and use this understanding to provide help and to update the student model.

- Intelligent analysis of student solutions: this deals with students’ answers with intelligent analysts. The intelligent analysts not only tell the student whether the solution is correct but also tell what exactly is wrong or incomplete and which missing or incorrect knowledge may be responsible for the error.

- Example-based problem solving support: this makes use of a learner’s earlier experience as examples that guide them to solve new problems.

- Adaptive collaboration support: this is to use the system’s knowledge about different users (stored in user models) to form a matching collaborative group.

In this paper, we mainly focus on learning and teaching activities using learning components rather than the design of course content structure or handling of communication infrastructure among several functions of web-based learning system.

3 An Overview of OO Software Development with UML

OO (Object-Oriented) analysis and design techniques were developed to help in the modelling of business, the analysis of requirements and the design software. With the rapid growth of OO technology, a number of techniques in OO have been developed and integrated. UML was designed to bring together the best feature of a number of analysis and design techniques and notations to produce an industry standard.

The Unified Modeling Language (UML) is a language for specifying, visualizing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. A software development process with UML consists of the following phases.

- Requirement Specification (Requirement Capture and Analysis): In this phase, the major purpose is to capture the requirements of the client to a system and to create a use-case model and a conceptual model.

- Design: This phase aims to create the interaction diagrams for the system operations and the design class diagrams.
Implementation: This is a process for mapping the artifacts created in the design stage to a code in real programming language.

3.1 Requirement Specification

In general, a requirement is a feature that the system must have or a constraint that it must satisfy to be accepted by the client. The requirements specification is about communication among developers, client, and users for defining a new system; it focuses on describing the purpose of the system and what to be done by the system. In an object-oriented development method it is recommended that the requirement specification at least includes the parts such as an overview of the project, the goals, the system functions, the system attributes, the use cases, and conceptual model.

3.2 Creation of Use Case Model

In object-oriented analysis, it is important to create and use use-case model. A use case describes a service provided by the system as a set of events that yields visible results for some users. Users are modeled by actors. Actors initiate a use case to access the system functionality. The use case can then initiate other use cases and gather more information from the actors. To identify use cases, read the existing requirements from an actor’s point of view and carry on discussions with those who will act as actors.

All the actors and use cases of a system make up a use-case model which describes how the use cases relate to each other and to the actors, and specifies the system functional requirements. A use case diagram describes part of the use case model.

3.3 Creation of a Conceptual Model

An important and typical activity in object-oriented requirement analysis is to identify concepts related to the requirements and to create a conceptual model of the domain. A conceptual model illustrates abstract and meaningful concepts in the problem domain. The creation of concepts is the most essential object-oriented step in analysis in investigation of the problem domain for building genuinely extensible software with reuse. The aim of this step is decomposition of the problem into individual concepts or objects.

For creation of conceptual model, the following notations in UML are used.

- **Concepts (Classes)**: A concept is represented by a class with a given name. An instance of a concept is called an object of the corresponding class. Therefore, a class defines a
set of objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.

- **Association**: A conceptual model with totally independent concepts only are obviously useless, as objects in different classes must be related to each other so that they can interact and collaborate with each other to carry out processes.

An association is a relationship between two classes that specifies how instances of the classes can be linked together to work together. As instances of a class are objects, instances of an association are links between objects of the two classes. In terms of an association between two classes, the notion of multiplicity indicates the information about how many objects of one class can be associated with one object of the other class, at a particular moment in time. On occasions it is necessary to clarify the role played by a class in an association. Therefore, each end of an association can be labeled by a string called role.

- **Aggregation**: An aggregation association is used to indicate that, as well as having attributes of its own, an instance of one class may consist of, or include, instances of another class. The actual number of ‘part’ instances will depend on the multiplicity at the part end of the association. The general concept of aggregation is often referred to as being a whole-part or part of relationship. One class is a part of another class.

- **Generalization**: A generalization is the relationship between a general class and one or more specialized classes. In a generalization relationship, the specializations are known as subclass and the generalized class is known as the superclass. A generalization allows the inheritance of the attributes and operations of a superclass by its subclasses.

- **Attributes of classes**: Each instance of a class may have some useful properties.

An attribute of a class is the abstraction of a single characteristic or a property of entities that have been abstracted as objects of the class. At any given moment of time, the attribute of an individual object in the class is a logical data value representing the corresponding property of the object, and called the value of attribute for the object at that time.

Class diagrams describe the static structure of the system in terms of classes and objects.

### 3.4 Creation of Interaction Diagram between Objects

A major task of the system design phase is to create the interaction diagrams for the system operations. The UML define two kinds of interaction diagrams, that is, collaboration diagram and object sequence diagram.

In object-oriented systems, the functionality that user requires to a system is produced by objects working together. Each individual object provides only a small element of the functionality -
its particular responsibilities - but when they work together, objects are able to produce high-level functionality that people can use. In order to work together in this way, objects need to communicate with each other, and they do this by passing messages. This ‘working together’ to produce some useful result is performed by collaboration between objects.

A collaboration diagram is a graph showing a number of objects and the links between them, which in addition shows the messages that are passed from one object to another. Collaboration diagrams can be used while the class diagram is being elaborated to help the analyst understand the grouping of the objects that participate in the realization of each use case. They can be used when the class diagram is more complete in order to understand and document the interactions among objects. They can also be used to specify the objects that take part in operations. The main purposes of producing collaboration diagrams are follows.

- They are used to model collaborations between objects or roles that deliver the functionality of a use case.
- They are used to model collaborations between objects or roles that deliver the functionality of an operation.
- They are used to model mechanisms within the architectural design of the system.
- They are annotated with interactions. These interactions show the messages that are passed between objects or roles within the collaboration.

A object sequence diagram is used to show the same interaction as in a collaboration diagram, but it emphasizes the order of the messages over time.

3.5 Mapping a Design Class Diagram to Code

The final goal of an object-oriented development of a system is the creation of code in an object-oriented programming language. The artifacts created in the design phase provide the information necessary in order to generate the code.

Mapping a design class diagram to a basic class definition in an object-oriented programming language involves the following steps:

- class definitions define the classes in the design class diagram in terms of the programming notation.
- method definitions - define the methods of classes in the design class diagram in terms of the programming notation.

These are derived from the interaction diagrams.
4 A WbLS system

This section shows an object-oriented development process of WbLS system with UML.

4.1 A Requirement Analysis of WbLS

We specified the requirements of WbLS system as follows.

Overview of the project

The overview of the WbLS project can be simply written as follows:

The purpose of this project is to create a web-based learning system for computer science subject to be used in universities or for distance students.

Goals

Based on the goals of a general education system described earlier, the goals of our WbLS can be stated as

- presentation of a curriculum on Computer Science through web browser,
- management of courses by adding, deleting, or updating course,
- scheduling of adaptive course offering for the registered courses,
- fast and exact delivery of course for distance students,
- offering several instructional components such as lecture descriptions, questions, answers, examinations, homeworks, and problems needed for students,
- checking or assessment on knowledge level of students,
- sequencing of learning content according to a student’s competency, objective, the result of the student’s interaction with learning activity, and sequencing rules,
- supporting of registration function of system users such as a student, a teacher, and a schedule manager.
- and delivery message between teachers and students through communication environment such as e-mail, NetMeeting, chat, or submitting by web browser.
System functions

System functions are what a system is supposed to do. The system’s function should be categorized in order to prioritize them or to avoid from missing them. The categories of functions are as follows.

- **Evident functions** should be performed by system, and user should be able to tell if they are performed.

- **Hidden functions** should be performed, but user need not be able to tell that they are performed. Hidden functions are often incorrectly missed during the requirements gathering process.

For the WbLS, we present four groups of the system functions: the basic functions and the functions related to particular actions. The basic functions are shown in Table 1.
Table 1: Basic functions of the WbLS system

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1</td>
<td>Record the detailed information (e.g. identifier, password, name, age, address, eMail address and so on) of a person (e.g. student or teacher) to the system.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.2</td>
<td>Display a list of courses on computer science subject.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.3</td>
<td>Find a course by input key (e.g. course title, teacher name, course comment, or duration) and display its details.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.4</td>
<td>Check the entered identifier and password of a user connected to the system in order to use the system and allow the registered user to login/out to the system.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.5</td>
<td>Display a list of courseofferings with their comments on the selected course.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.6</td>
<td>Display description and learning components of the selected course for the registered person.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.7</td>
<td>Find a courseoffering selected by a student or a teacher and display its details.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.8</td>
<td>Display a list of several learning components on the selected courseoffering (For example, lesson list, questionnaire, homework sheet, or examination sheet.).</td>
<td>evident</td>
</tr>
<tr>
<td>R1.9</td>
<td>Find content description of a lesson or problems of an exam sheet by input information of a user.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.10</td>
<td>Display the assessment of the outcome of students’ interaction with learning activity (For example, assessment of students’ examination).</td>
<td>evident</td>
</tr>
<tr>
<td>R1.11</td>
<td>Handle the feedback between teachers and students by transferring messages for several learning activities.</td>
<td>evident</td>
</tr>
<tr>
<td>R1.12</td>
<td>Update the outcome of student’s learning activities on a course taken.</td>
<td>hidden</td>
</tr>
<tr>
<td>R1.13</td>
<td>According to the outcome of student’s learning activity on a course taken, update the conditions of sequencing.</td>
<td>hidden</td>
</tr>
<tr>
<td>R1.14</td>
<td>According to the sequencing conditions, sequence learning component such as lessons and exam sheets.</td>
<td>hidden</td>
</tr>
<tr>
<td>R1.15</td>
<td>Handle communication environments such as e-mail, NetMeeting, chat, or submitting of web browser.</td>
<td>hidden</td>
</tr>
</tbody>
</table>

We identified respectively the other individualized functions excepting for the basic functions. Those are given in from Table 2 to Table 4.
### Table 2: Functions for students only

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2.1</td>
<td>Register a student for a course, and display registration information of the student for the course if a student was registered to the course.</td>
<td>evident</td>
</tr>
<tr>
<td>R2.2</td>
<td>Update learning profile of the student on the lesson activity after a student took a lesson.</td>
<td>hidden</td>
</tr>
<tr>
<td>R2.3</td>
<td>Handle group works with the other students, using communication environments such as e-mail, NetMeeting, or chat provided.</td>
<td>evident</td>
</tr>
<tr>
<td>R2.4</td>
<td>Display learning profile of a student.</td>
<td>evident</td>
</tr>
</tbody>
</table>

### Table 3: Functions for teachers only

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3.1</td>
<td>Display the registered students’ list and their learning profiles on a courseoffering.</td>
<td>evident</td>
</tr>
<tr>
<td>R3.2</td>
<td>For all student registered in a course, display totally mark table of learning components such as questionnaire, homework, or examination and collect their statistics.</td>
<td>evident</td>
</tr>
<tr>
<td>R3.3</td>
<td>Assess students’ solutions for a examination or a homework.</td>
<td>evident</td>
</tr>
<tr>
<td>R3.4</td>
<td>Update the outcomes of learning activities of students who are taking a courseoffering currently.</td>
<td>evident</td>
</tr>
<tr>
<td>R3.5</td>
<td>Handle course database, submitting content description of new lesson to database server according to scheduling.</td>
<td>evident</td>
</tr>
<tr>
<td>R3.6</td>
<td>Handle course database, submitting description of new learning components such as examination and homework to database server according to scheduling.</td>
<td>evident</td>
</tr>
<tr>
<td>R3.7</td>
<td>Add new course to or delete a course from course database.</td>
<td>evident</td>
</tr>
</tbody>
</table>

### Table 4: Functions for scheduling (for schedule managers)

<table>
<thead>
<tr>
<th>Ref #</th>
<th>Function</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4.1</td>
<td>Display a list of teachers qualified to a course and details of the course.</td>
<td>evident</td>
</tr>
<tr>
<td>R4.2</td>
<td>Schedule and display the courseoffering of a course.</td>
<td>evident</td>
</tr>
<tr>
<td>R4.3</td>
<td>Add a new courseoffering, modify the date of a course, or cancel a unnecessary courseoffering based on students’ learning profile.</td>
<td>evident</td>
</tr>
</tbody>
</table>
4.2 Use Cases and Use Case Diagrams

Describing Use Cases and Actors

In order to describe use cases and actors of this system, we first considered the roles of several users or external systems using the system as follows.

- **Students** must register their information to the system, and their learning profile to the course for taking a course. They are also allowed to view the posted course notes and assignments, take the online questions, examinations, and homeworks, and complete and submit the evaluation forms for the teacher.

- **Teachers** are responsible for adding a new course, creating lecture notes, examinations, homeworks and evaluation forms, presenting answers corresponding to students' questions, viewing their results, and assessing the results of students' solutions. They must also register their information to the system in order to use the system.

- **System administrator** is granted the privileges to add a new teacher or a new student to the database, register them for a particular course, query the list of students registered for any particular course, or the list of courses lectured by a particular teachers, etc.

- **Schedule Manager** is responsible for organizing the schedule on a course.

- **Learning Management System** (LMS) is responsible for sequencing learning components such as lesson or examination according to sequencing rules and students' learning profiles.

Therefore, we identified the **Actors** such as **Student**, **Teacher**, **Schedule Manager**, **LMS System**, and **System Administrator**, as external entities that interact with the system.

On the other hand, **Use Cases** describe the behavior of the system as seen from an actor's point of view.

In our system we considered the **Take Course Offering with Lesson** use case for **Student** actor.

We first created a high-level use case to obtain some understanding of the overall process, and then expanded it by adding to it with more details. In general, to describe a **high-level use case**, a template composed of several fields such as **name of use case**, **participating actors**, **purpose**, **pre-condition**, **overview**, **post-condition** and **cross references** is used.

Therefore, the high-level description of the **Take Course Offering with Lesson** use case can be described as follows.
Use case : Take Courseoffering with Lesson

Actors : Student

Purpose : Increase student’s knowledge on Computer Science by taking courseoffering in the WbLS.

Pre-Condition : A Student must already be registered on the course related to the courseoffering in order to take a courseoffering.

Overview : The student registered to the system runs WbLS system through web browser and selects a course. The student selects a suitable courseoffering from a list of courseofferings related to the course. The student selects lecture notes among several learning components of the selected courseoffering. She selects a lesson corresponding to her objective from a list of lessons delivered. She browses the content descriptions delivered for the selected lesson. According to the result of the student’s interaction with the lesson, the learning profile of the student is updated.

Post-Condition : For the student, the learning profile such as learning activity of the lesson is updated. The knowledge on Computer Science taken through this system has been added to the student’s accomplishment.

Cross References : Functions: R1.1, R1.2, R1.3, R1.4, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10, R1.12, R3.1, R3.2, R3.3, R3.4

The references to the system functions indicate that the use case is created through further understanding of these functions and these required functions are allocated to this use case.

An expanded use case shows more details than a high-level one, and is often done in a conversational style between the actors and system. Typically, an expanded use case extends a high-level one with two sections typical course of events and alternative courses of events (or exceptions).

The typical flow of events of Take Courseoffering with Lesson use case is shown as follows.
Flow of events for Taking Courseoffering with Lesson

Typical Course of Events

**Actor Action** | **System Response**
--- | ---
1. This use case begins when a student takes a course offering using WbLS through the web browser. The student inputs identifier registered to the system, and the title of a course. | 2. For a registered student, the system finds the course corresponding to the entered course’s title from course database and displays a list of course offerings related to the course.

3. The student selects a course offering from the list. | 4. The system displays several learning components such as Lecture Notes, Homework, Examination, and Questionnaire on the selected course offering.

5. The student selects the 'Lecture Notes' option from learning components. | 6. Based on the sequencing rule and the student’s learning profile, the system sequences and displays lessons on selected course offering.

7. The student selects a lesson from the list. | 8. The system finds content description of the lesson that the student required from course database and displays. The system delivers learning content based on the status of lesson taken by the student.

9. Using navigation, the student browses the content description displayed on browser. After satisfied learning objective, the student submits "End" request to the system and exits from the system. | 10. According to the result of the student’s interaction with the lesson, the system updates the student’s learning profile on the course offering and the lesson.

11. The student finishes taking the course offering.

**Alternative Courses**

- Line 1: If the student was unregistered to the course, indicate error.

- Line 7: According to the status of the lesson taken by the student, indicate warning message.
The sequence of actions from 5 to 10 in the **Take Courseoffering with Lesson** use case can be treated as a use case, which we can call **Take Lesson**. This **Take Lesson** use case can be simply written as:

---

**Flow of events for Take Lesson Use Case**

**Typical Course of Events**

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The student selects the 'Lecture Notes’ option from several learning components on the selected courseoffering.</td>
<td>2. Based on the sequencing rule and the student’s learning profile, the system sequences and displays lessons on selected courseoffering.</td>
</tr>
<tr>
<td>3. The student selects a lesson from the list.</td>
<td>4. The system finds content description of the lesson that the student required from course database and display. The system delivers learning content based on the status of lesson taken by the student.</td>
</tr>
<tr>
<td>5. Using navigation, the student browses the content description displayed on browser. After satisfied learning objective, the student submits ”End” request to the system and exits from the system.</td>
<td>6. According to the result of the student’s interaction with the lesson, the system updates the student’s learning profile on the courseoffering and the lesson.</td>
</tr>
</tbody>
</table>

Using the same techniques in the creation of **Take Lesson** use case, we can create three use cases **Take Homework**, **Take Examination**, and **Ask Question**.

Finally we can update the flow of events of the general use case **Take Courseoffering** as follows.
Flow of events for Take Courseoffering Use Case

Typical Course of Events

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. This use case begins when a student takes a courseoffering using WbLS through the web browser. The student inputs identifier registered to the system, and the title of a course.</td>
<td>2. For a registered student, the system finds the course corresponding to the entered course’s title from course database and displays a list of courseofferings related to the course.</td>
</tr>
<tr>
<td>3. The student selects a courseoffering from the list.</td>
<td>4. The system displays several learning components such as Lecture Note, Homework, Examination, and Questionnaire on the selected courseoffering.</td>
</tr>
<tr>
<td>5. The student selects a component from several learning components on the selected courseoffering. (a) If Lesson, initiate Take Lesson. (b) If Question, initiate Ask Question. (c) If Homework, initiate Take Homework. (d) If Examination, initiate Take Examination.</td>
<td>6. According to the result of the student’s interaction with the component, the system updates the student’s learning profile on the courseoffering and the component.</td>
</tr>
<tr>
<td>7. The student finishes taking the courseoffering.</td>
<td></td>
</tr>
</tbody>
</table>

Alternative Courses
- Line 1: If the student was unregistered to the course, indicate error.

So far, we considered only process describing the Take Courseoffering use case for Student actor as a sample. For all use cases of the system, these descriptions can be applied.
Use case diagrams

Use case diagrams show use cases and actors and the external interactions between them as a graphic model. Use case diagrams are supported by behavior specifications, which define the interactions within a particular use case. In general, it is often used the relationships between use cases such as include and extend to reduce the complexity of the model by identifying commonalities in different use cases in describing use case diagrams.

A include relationship can be used for making up a big use case from simpler ones. A extend relationship is used in occasions where one use case may optionally be extended by the functionality in another use case.

Several use case diagrams are shown in the following Figures 3-7. As shown in the figures, we used the include relationship between use cases to describe use case diagrams.

Figure 3. Use Case Diagram by the Student actor
A WbLS system

Figure 4. Use Case Diagram by the Teacher actor

Figure 5. Use Case Diagram by the ScheduleManager actor
A list of relevant actors and use cases considered in our system is shown in the following table.
Table 5: Actors and Use cases for WbLS

<table>
<thead>
<tr>
<th>Actor</th>
<th>Use case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Login/out</td>
</tr>
<tr>
<td></td>
<td>Register with System</td>
</tr>
<tr>
<td></td>
<td>Register to Course</td>
</tr>
<tr>
<td></td>
<td>Take Courseoffering</td>
</tr>
<tr>
<td></td>
<td>&amp;include;  : Take Lesson, Ask Question,</td>
</tr>
<tr>
<td></td>
<td>  Take Homework, Take Examination</td>
</tr>
<tr>
<td>Teacher</td>
<td>Login/out</td>
</tr>
<tr>
<td></td>
<td>Register with System</td>
</tr>
<tr>
<td></td>
<td>Add Course</td>
</tr>
<tr>
<td></td>
<td>Update Course</td>
</tr>
<tr>
<td></td>
<td>&amp;include;  : Create/Update Lesson,</td>
</tr>
<tr>
<td></td>
<td>  Answer Question,</td>
</tr>
<tr>
<td></td>
<td>  Create/Update Homework,</td>
</tr>
<tr>
<td></td>
<td>  Create/Update Examination</td>
</tr>
<tr>
<td></td>
<td>Delete Course</td>
</tr>
<tr>
<td>Schedule Manager</td>
<td>Schedule Courseoffering</td>
</tr>
<tr>
<td></td>
<td>&amp;include;  : Add Courseoffering,</td>
</tr>
<tr>
<td></td>
<td>  Delete Courseoffering,</td>
</tr>
<tr>
<td></td>
<td>  Update Courseoffering</td>
</tr>
<tr>
<td>LMS System</td>
<td>Sequence Course</td>
</tr>
<tr>
<td></td>
<td>&amp;include;  : Sequence Lesson,</td>
</tr>
<tr>
<td></td>
<td>  Sequence Exam</td>
</tr>
<tr>
<td>System Administrator</td>
<td>Get List of all the Students</td>
</tr>
<tr>
<td></td>
<td>Get List of all the Teachers</td>
</tr>
<tr>
<td></td>
<td>Get List of Courses</td>
</tr>
<tr>
<td></td>
<td>Add/Update Teacher Information</td>
</tr>
<tr>
<td></td>
<td>Add/Update Student Information</td>
</tr>
<tr>
<td></td>
<td>Add/Update Course Details</td>
</tr>
</tbody>
</table>

4.3 Conceptual Model

Concepts (Classes)

We can identify the concepts of the domain of this project based on previously specified use case model. In an object-oriented development, there are usually two strategies to identify concepts. The first is to find concepts from the textual descriptions of the problem domain according to a concept category list. This strategy is to create a list of candidate concepts from the client’s requirements description, initial investigation reports, system functions definitions, and use cases. Another useful and simple technique for identification of concepts is to identify the noun and noun phrases in the textual descriptions of a problem domain, and consider them as candidate concepts or attributes.
We identified concepts in problem domain of WbLS as follows.

<table>
<thead>
<tr>
<th>Component</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>CourseController</td>
<td>Answer</td>
</tr>
<tr>
<td>CourseoffController</td>
<td>Solution</td>
</tr>
<tr>
<td>Person</td>
<td>OverallSequencing</td>
</tr>
<tr>
<td>Student</td>
<td>Activity</td>
</tr>
<tr>
<td>Teacher</td>
<td>Rule</td>
</tr>
<tr>
<td>ScheduleManager</td>
<td>LMS</td>
</tr>
<tr>
<td>Subject</td>
<td>Computer Science</td>
</tr>
<tr>
<td>Course</td>
<td>CourseOffering</td>
</tr>
<tr>
<td>Registration</td>
<td>SystemAdministrator</td>
</tr>
<tr>
<td>SystemAdministrator</td>
<td>LearningUnit</td>
</tr>
<tr>
<td>Component</td>
<td>LectureNotes</td>
</tr>
<tr>
<td>Questionnaire</td>
<td>Homework</td>
</tr>
<tr>
<td>Homework</td>
<td>Examination</td>
</tr>
<tr>
<td>LoginRecord</td>
<td></td>
</tr>
<tr>
<td>Lesson</td>
<td>HomeworkSheet</td>
</tr>
<tr>
<td>ExamSheet</td>
<td>Question</td>
</tr>
<tr>
<td>Questionnaire</td>
<td>HomeworkProblem</td>
</tr>
<tr>
<td>ExamProblem</td>
<td>Assessmen</td>
</tr>
<tr>
<td>Content</td>
<td>Problem</td>
</tr>
<tr>
<td>Answer</td>
<td></td>
</tr>
<tr>
<td>Solution</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8. Concepts in WbLS**

**Association**

The identified concepts can be associated with each other, as objects in different concepts (classes) must be related to each other so that they can interact and collaborate with each other to carry out processes in the system. For example, a student is associated with the course for registering to a course and a course is associated with a course offering scheduled. On the other hand, a teacher can be qualified to a number of courses, a student can be registered to several courses, and a course may contain several course offerings. Such an information can be applied to the identified concepts.

Using an association between two classes and the notation of multiplicity in UML, we extended a number of concepts in Figure 8 to a conceptual model with associations (Figure 9).
Aggregation

An aggregation association is used to indicate that, as well as having attributes of its own, an instance of one class may consist of, or include, instances of another class. The actual number of ‘part’ instances will depend on the multiplicity at the part end of the association. Figure 10 shows an aggregation relationship between several classes identified in this system. This system consists of a number of subjects and Computer Science subject is one part of the subjects. Also Computer Science subject can contain a number of courses. A course, in turn, includes a number of courseofferings according to the starting date of a course, a courseoffering contains several learning units such as lessons, exam sheets, and homework sheets.
Figure 10. Aggregation

Generalization

A generalization is the relationship between a general class and one or more specialized classes. In a generalization relationship, the specializations are known as subclass and the generalized class is known as the superclass. A generalization allows the inheritance of the attributes and operations of a superclass by its subclasses. In Figure 11 there are examples of generalization relationship described in this system. Here Person, Component, and LearningUnit are superclasses respectively, and other classes are all subclasses.
Attributes of classes

Each instance of a class may have some useful properties. An attribute of a class is the abstraction of a single characteristic or a property of entities that have been abstracted as objects of the class. At any given moment of time, the attribute of an individual object in the class is a logical data value representing the corresponding property of the object, and called the value of attribute for the object at that time.

We added a number of attributes to classes identified for this system as follows.
By combining the Figure 9, Figure 10, Figure 11, and Figure 12, we can obtain the conceptual model for a comprehensive consideration of WbLS as shown in Figure 13.
Figure 13. A Conceptual Model for WbLS
4.4 A Behavior of WbLS System

System design is the transformation of the analysis model into a system design model. During requirements capture and analysis, we concentrated on the purpose and the functionality of the system, but during system design, we focus on the processes, data structures, and software hardware components necessary to implement it.

System Sequence Diagrams

Before getting to the design of the system, we first need to analyze the use cases and identify the system operations and the order in which they are performed to carry out a use case. We do this by treating the system as a black box. We use a system sequence diagram or a trace diagram to show the interaction between the actors and the system.

For example, the typical course of events for selecting a course offering by a registered student indicates that the student generates the system input events that can be denoted as login, registerCourse, and selectCourseoff. This course is shown as a system sequence diagram in Figure 14.

![System Sequence Diagram](image)

**Figure 14.** A system sequence diagram for **Take Courseoffering** use case

Recoding System Operation

The set of all required system operation is determined by identifying the system input events. Each system input event causes the execution are identical; the distinction is that the input event is the named stimulus, the operation is the response.

We identified the following system operations for WbLS.
<table>
<thead>
<tr>
<th>Use case</th>
<th>Actor</th>
<th>System Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login/out</td>
<td>Student, Teacher</td>
<td>login()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>logout()</td>
</tr>
<tr>
<td>Register with system</td>
<td>Student, Teacher</td>
<td>registerSystem()</td>
</tr>
<tr>
<td>Register to Course</td>
<td>Student</td>
<td>registerCourse()</td>
</tr>
<tr>
<td>Take Courseoffering</td>
<td>Student</td>
<td>selectCourseoff()</td>
</tr>
<tr>
<td>Take Lesson</td>
<td>Student</td>
<td>selectLesson()</td>
</tr>
<tr>
<td>Take Examination</td>
<td>Student</td>
<td>selectEmSheet()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>makeEmSolution()</td>
</tr>
<tr>
<td>Ask Question</td>
<td>Student</td>
<td>makeQuestion()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>displayAnswer()</td>
</tr>
<tr>
<td>Take Homework</td>
<td>Student</td>
<td>selectHmSheet()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>makeHmSolution()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>endHomework()</td>
</tr>
<tr>
<td>Add Course</td>
<td>Teacher</td>
<td>addCourse()</td>
</tr>
<tr>
<td>Create/Update Lesson</td>
<td>Teacher</td>
<td>addLesson()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updateLesson()</td>
</tr>
<tr>
<td>Create/Update Examination</td>
<td>Teacher</td>
<td>addEmSheet()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>assessResult()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>displayAssessment()</td>
</tr>
<tr>
<td>Create/Update Homework</td>
<td>Teacher</td>
<td>addHmSheet()</td>
</tr>
<tr>
<td>Answer Question</td>
<td>Teacher</td>
<td>makeAnswer()</td>
</tr>
<tr>
<td>Add Courseoffering</td>
<td>ScheduleManager</td>
<td>addCourseoff()</td>
</tr>
<tr>
<td>Delete Courseoffering</td>
<td>ScheduleManager</td>
<td>deleteCourseoff()</td>
</tr>
<tr>
<td>Update Courseoffering</td>
<td>ScheduleManager</td>
<td>updateCourseoff()</td>
</tr>
<tr>
<td>Sequence Course</td>
<td>LMS System</td>
<td>updateCourseActivity()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sequenceCourse()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updateSeqCondition()</td>
</tr>
<tr>
<td>Sequence Lesson</td>
<td>LMS System</td>
<td>sequenceLesson()</td>
</tr>
<tr>
<td>Sequence Exam</td>
<td>LMS System</td>
<td>sequenceExam()</td>
</tr>
</tbody>
</table>

We omitted the parameters in the above table for recoding system operations. The UML includes notation to record operations of a class. With this notation, the system operation can be grouped as operations of a class called *controller class* shown later on.
Contracts for System Operations

The contracts of each system operation are shown below.

**Contract for login**

**Name:** login(userID : string, password : string).

**Responsibilities:** Check the user with the userID and password from person database. For the registered user, record login information of the user.

**Type:** System.

**Cross References:** System Function : R1.4  
Use Cases : Login/out

**Note:** Use relational database access such as SQL, Oracle.

**Exception:** If the user is not registered or the entered value is wrong, indicate an error message.

**Output:** The userID and password of a user are known to the system.

**Pre-conditions:**

- A LoginRecord was created (instance creation).

- The new LoginRecord was associated with the WbLS (association formed).

- The LoginRecord.LoginDate was set to current date (attribute modification).

- The LoginRecord.LoginTime was set to current time (attribute modification).

- The Person.Status was set to Active (attribute modification).
**Contract for logout**

**Name:**
logout(userID : string)

**Responsibilities:**
Disconnect from the system the user with the userID.

**Type:**
System.

**Cross References:**
System Function : R1.4
Use Cases : Login/out

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**
If disconnection from the system fail, indicate an error message.

**Output:**
The userID of a user is known to the system.

**Pre-conditions:**
The userID of a user is known to the system.

**Post-conditions:**
- The LoginRecord.LogoutDate was set to current date (attribute modification).
- The LoginRecord.LogoutTime was set to current time (attribute modification).
- The Person.Status was set to Inactive (attribute modification).
Contract for registerSystem

Contract

Name : registerSystem(userID : string, pass : string, type : string, name : string, sex : string, age : int, address : string, eMailAddress : string)

Responsibilities : If the entered information is valid, record the user with the information to person database.

Type : System.

Cross References : System Function : R1.1
Use Cases : Register with System

Note : Use relational database access such as SQL, Oracle.

Exception : If the entered information is incorrect, indicate an error message and cancel recording.

Output :

Pre-conditions : The userID, password, type, and details of a user such as name, age, address and eMailAddress are known to the system.

Post-conditions :

- If the user a student, a Student was created (instance creation).
- If the user a student, the new Student was associated with the WbLS (association formed).
- If the user a teacher, a Teacher was created (instance creation).
- If the user a teacher, the new Teacher was associated with the WbLS (association formed).
Contract for \textit{registerCourse} \\

\textbf{Name :} \quad \text{registerCourse}(stID : \text{string}, coTitle : \text{string}) \\
\textbf{Responsibilities :} \\
Find the course by the entered title from course database. Display details of the course such as course description and the qualified teacher information. Create new registration for the student and add learning details of the student (e.g., competency and learning objective). \\
\textbf{Type :} \\
System. \\
\textbf{Cross References :} \\
System Function : R1.3, R2.1 \\
Use Cases : Register to Course \\
\textbf{Note :} \\
Use relational database access such as SQL, Oracle. \\
\textbf{Exception :} \\
If the title of a course is not found, indicate a message that can not find those. \\
If the registered prerequisite of the student is different from the prerequisite of the course, indicate a message that the condition for registering is not satisfied. \\
\textbf{Output :} \\
\textbf{Pre-conditions :} \\
The identifier of a student and the title of a course are known to the system. \\
\textbf{Post-conditions :} \\
\begin{itemize} \\
  \item A new Registration of the student was created (instance creation). \\
  \item The new Registration was associated with the WbLS (association formed). \\
  \item The Registration.Date was set to current date (attribute modification). \\
\end{itemize}
**Contract for selectCourseoff**

**Contract**

**Name:** selectCourseoff(stID : string, coTitle : string, startDate : string)

**Responsibilities:** Find the courseoffering with the entered startDate from a list of courseofferings related to the course with coTitle. Display details of the courseoffering and a list of several learning components.

**Type:** System.

**Cross References:**
- System Function : R1.3, R1.5, R1.6, R1.7, R1.8
- Use Cases : Take Courseoffering

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**
- If the current date is not between startDate and endDate of the selected courseoffering, indicate a message that can not take the courseoffering.

**Output:**

**Pre-conditions:**
- The identifier of a student, information of a course, and the title of a courseoffering are known to the system.

**Post-conditions:**

- If a new activity, a new Activity on the learning activity of the student with stID with was created (instance creation).
- The new Activity was associated with the CourseController (association formed).
- The Activity.IsAttempted was set to True (attribute modification).
**Contract for selectLesson**

**Contract**

**Name:**
selectLesson(stID : string, coTitle : string, startDate : string)

**Responsibilities:**
According to the objective and the result of learning activity of the student, display a list of lessons on the selected courseoffering.

**Type:**
System.

**Cross References:**
System Function : R1.8, R1.9, R2.2
Use Cases : Take Lesson, Create/Update Lesson

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The title of a course, the title of a courseoffering and the title of a lesson are known to the system.

**Post-conditions:**
- If a new activity, a new Activity on the leaning activity of the student with stID with was created (instance creation).
- The new Activity was associated with the Courseoff-Controller (association formed).
- The Activity.IsAttempted was set to True(attribute modification).

**Contract for endLesson**

**Contract**

**Name:**
endLesson(stID : string, coTitle : string, startDate : string)

**Responsibilities:**
For the selected lesson of the course with coTitle, record the completed information of a student to course database.

**Type:**
System.

**Cross References:**
System Function : R2.2
Use Cases : Take Lesson

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The title of a course, the title of a courseoffering and the title of a lesson are known to the system.

**Post-conditions:**
- The Activity.IsCompleted was set to True(attribute modification).
Contract for *selectEmSheet*

**Name:**
selectEmSheet(stID : string, coTitle : string, startDate : string)

**Responsibilities:**
According to the objective and the result of learning activity of the student, display a list of a corresponding examination sheets on the selected courseoffering.

**Type:**
System.

**Cross References:**
System Function : R1.9, R2.2
Use Cases : Take Examination, Create/Update Examination
Use relational database access such as SQL, Oracle.

**Note:**
The title of a course, the title of a courseoffering and the title of an examination sheet are known to the system.

**Exception:**

**Output:**

**Pre-conditions:**
The title of a course, the title of a courseoffering and the title of an examination sheet are known to the system.

**Post-conditions:**

- If a new activity, a new Activity on the leaning activity of the student with stID with was created (instance creation).
- The new Activity was associated with the selected ExamSheet (association formed).
- The Activity.IsAttempted was set to True(attribute modification).
**Contract for makeEmSolution**

**Name:**
makeEmSolution(stID : string, coTitle : string, startDate : string)

**Responsibilities:**
Add a new solution created on problems of the selected examination sheet.

**Type:**
System.

**Cross References:**
System Function : R1.9, R1.10, R1.11, R1.15
Use Cases : Take Examination, Create/Update Examination

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The identifier of a student, the title of a course, the title of a courseoffering, the title of an examination sheet, and exam problem are known to the system.

**Post-conditions:**

- A new Solution was created (instance creation).
- The new Solution was associated with the EmSheet (association formed).

---

**Contract for makeQuestion**

**Name:**
makeQuestion(stID : string, coTitle : string, startDate : string)

**Responsibilities:**
Add a new question to questionnaire of a course database.

**Type:**
System.

**Cross References:**
System Function : R1.9, R1.10, R1.11, R1.15
Use Cases : Ask Question

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The identifier of a student, the title of a course, and the title of a courseoffering are known to the system.

**Post-conditions:**

- A new Question was created (instance creation)
- The new Question was associated with the Courseoffering (association formed).
- The Question.Question was set (attribute modification).
Contract for \textit{addCourse}

\textbf{Name :} \texttt{addCourse(teID : string, coTitle : string, comment : string)}

\textbf{Responsibilities :} Add the information of a new course to course database.

\textbf{Type :} System.

\textbf{Cross References :} System Function : R3.7 Use Cases : Add Course

\textbf{Note :} Use relational database access such as SQL, Oracle.

\textbf{Exception :}

\textbf{Output :}

\textbf{Pre-conditions :} The identifier of a teacher and the details of a course are known to the system.

\textbf{Post-conditions :}

- A new Course was created (instance creation).
- The new Course was associated with the Computer-Science (association formed).
- The ComputerScience.CourseCount was increased (attribute modification).

---

Contract for \textit{addLesson}

\textbf{Name :} \texttt{addLesson(teID : string, coTitle : string)}

\textbf{Responsibilities :} Add the information of the entered lesson with its content to course database.

\textbf{Type :} System.

\textbf{Cross References :} System Function : R3.6 Use Cases : Create/Update Lesson

\textbf{Note :} Use relational database access such as SQL, Oracle.

\textbf{Exception :}

\textbf{Output :}

\textbf{Pre-conditions :} The identifier of a teacher and the title of a course are known to the system.

\textbf{Post-conditions :}

- A new Lesson was created (instance creation).
- The new Lesson was associated with the LectureNote of the course with coTitle (association formed).
- The LectureNote.Count was increased (attribute modification).
**Contract for updateLesson**

**Contract**

**Name:** updateLesson(teID : string, coTitle : string, lessTitle : string, content : string)

**Responsibilities:** Update the content of the entered lesson with its content to course database.

**Type:** System.

**Cross References:** System Function : R3.6

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:** The identifier of a teacher, the title of a course, the title of a lesson, and content of the lesson are known to the system.

**Post-conditions:** The Lesson.Content was modified (attribute modification).

---

**Contract for addEmSheet**

**Contract**

**Name:** addEmSheet(teID : string, coTitle : string)

**Responsibilities:** Add a new examination sheet with a number of exam problems to database.

**Type:** System.

**Cross References:** System Function : R1.6, R3.6

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:** The title of a course, and the teacher’s ID are known to the system.

**Post-conditions:**

- A new ExamSheet was created (instance creation).
- The new ExamSheet was associated with the selected Examination component (association formed).
- A new ExamProblem was created (instance creation).
- The new ExamProblem was associated with ExamSheet (association formed).
- The Examination.Count was increased (attribute modification).
Contract for assessResult

Name:    
Responsibilities:    
Record the assessment of a student’s solution on an exam sheet.

Type: System.

Cross References: System Function: R3.4 Use Cases: Create/Update Examination

Note: Use relational database access such as SQL, Oracle.

Exception: 
Output: The title of a course, the startDate of a course offering, and the title of exam sheet are known to the system.

Pre-conditions:

Post-conditions:

- A new Assessment was created (instance creation).
- The new Assessment was associated with Solution (association formed).
- The Assessment.Mark was set (attribute modification).
- The Assessment.Comment was set (attribute modification).
- The Activity.IsPassed was set to True or False (attribute modification).
Contract for *makeAnswer*

**Name:**
makeAnswer(teID : string, coTitle : string, startDate : string)

**Responsibilities:**
Make the answers corresponding to questions asked by students.

**Type:**
System.

**Cross References:**
System Function : R1.11
Use Cases : Answer Question

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The identifier of a teacher, the title of a course, and the start-
Date of a course offering are known to the system.

**Post-conditions:**

- A new Answer was created (instance creation).
- The new Answer was associated with Question (association formed).

Contract for *addCourseoff*

**Name:**
addCourseoff(teID : string, coID : string, startDate : string)

**Responsibilities:**
Add the entered course offering information to course offering table corresponding to the course with coID in course database.

**Type:**
System.

**Cross References:**
System Function : R4.2, R4.3
Use Cases : Add Course offering

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The title of a course and the start date of a course offering are known to the system.

**Post-conditions:**

- A new Courseoffering was created (instance creation).
- The new Courseoffering was associated with the Course with coID (association formed).
- The Courseoffering.EndDate was set by the duration of the course and the entered startDate (attribute modification).
Contract for *deleteCourseoff*

**Contract**

**Name:** deleteCourseoff(coID : string, offTitle : string)

**Responsibilities:** Delete a record of the courseoffering with offID among courseoffering of the course with coID from course database.

**Type:** System.

**Cross References:** System Function : R4.3

**Use Cases:** Delete Courseoffering

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:** The title of a course and the identifier of a courseoffering are known to the system.

**Post-conditions:** The CourseOfferingCount was decreased (attribute modification).

---

Contract for *updateCourseoff*

**Contract**

**Name:** updateCourseoff(coID : string, offTitle : string, startDate : date)

**Responsibilities:** Change a starting date of the courseoffering with offID among courseoffering of the course with coID from course database.

**Type:** System.

**Cross References:** System Function : R4.3

**Use Cases:** Delete Courseoffering

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:** The title of a course and the identifier of a courseoffering are known to the system.

**Post-conditions:**

- The CourseOffering.StartDate was changed (attribute modification).
- The CourseOffering.EndDate was changed (attribute modification).
Contract for *updateCourseActivity*

**Name:** updateCourseActivity(stID : string, coTitle : string)

**Responsibilities:** For the course with coTitle, find the information of taking the course activity of a student and update.

**Type:** System.

**Cross References:** System Function : R1.12
Use Cases : Sequence Course

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:** The identifier of a student and the title of a course are known to the system.

**Post-conditions:**

- The Activity.IsCompleted was set to True(attribute modification).
- The Student.Prerequisite was set to the course with coTitle (attribute modification).

Contract for *sequenceCourse*

**Name:** sequenceCourse(stID : string)

**Responsibilities:** According to learning competence and learning result of a student, sequence courses for the student with stID.

**Type:** System.

**Cross References:** System Function : R1.12, R1.14
Use Cases : Sequence Course

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:** The identifier of a student are known to the system.

**Post-conditions:**

- A new DeliverableCourse was created (instance creation).
- The new DeliverableCourse was associated with the OverallSequencing (association formed).
Contract for **updateSeqCondition**

**Name:**
updateSeqCondition(stID : string, coID : string)

**Responsibilities:**
Find activity information of a student on the course with coID. According to sequencing rule and learning result of the student with stID, update conditions for sequencing learning units.

**Type:**
System.

**Cross References:**
System Function : R1.13
Use Cases : Sequence Course

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The identifier of a student and the title of a course are known to the system.

**Post-conditions:**
The Activity.Objective was modified (attribute modification).

Contract for **sequenceLesson**

**Name:**
sequenceLesson(stID : string, coTitle : string)

**Responsibilities:**
According to learning result of a student, sequence lessons for the student with stID.

**Type:**
System.

**Cross References:**
System Function : R1.14
Use Cases : Sequence Lesson

**Note:**
Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:**
The identifier of a student and the title of a course are known to the system.

**Post-conditions:**

- A new DeliverableLesson was created (instance creation).
- The new DeliverableLesson was associated with the OverallSequencing (association formed).
A WbLS system

Contract for *sequenceExam*

**Name:** sequenceExam(stID : string, coTitle : string)

**Responsibilities:** According to learning result of a student, sequence exam sheets for the student with stID.

**Type:** System.

**Cross References:** System Function : R1.14
Use Cases : Sequence Exam

**Note:** Use relational database access such as SQL, Oracle.

**Exception:**

**Output:**

**Pre-conditions:** The identifier of a student and the title of a course are known to the system.

**Post-conditions:**

- A new DeliverableEmSheet was created (instance creation).
- The new DeliverableEmSheet was associated with the OverallSequencing (association formed).

4.5 A WbLS Design

4.5.1 Collaboration Diagrams

Collaboration diagrams can be produced via the following steps.

- Decide on the context of the interaction: system, subsystem, use case or operation.
- Identify the structural elements (class roles, objects, subsystems) necessary to carry out the functionality of this collaboration.
- Model the structural relationship between those elements to produce a diagram showing the context of the interaction.
- Consider the alternative scenarios that may be required.
- Draw instance level collaboration diagram, if required.
- Optionally draw a specification level collaboration diagram to summarize the alternative scenarios in the instance sequence diagrams.

In the UML, the GRASP (General Responsibility Assignment Software Pattern) patterns are used to assign responsibilities to class and to create collaboration diagram. These pattern consist
of the following five patterns: *Expert*, *Creator*, *Low Coupling*, *High Cohesion*, and *Controller*. In above section, system operations to be used in system design stage and the contracts for the system operations have already created.

In addition, we identified *CourseController* and *CourseoffController* controller classes, and added our conceptual model as follows.

![Controller Classes in Conceptual Model](image)

**Figure 15.** Controller Classes in Conceptual Model

Using system operations, contracts and GRASP patterns, we created a number of collaboration diagrams for each system operation below.

**Collaboration diagram for login**

*WbLS* is the controller for handling this operation. The responsibility in the contract for *login* indicates the need to find a user. *WbLS* is a candidate for finding those from multiple objects. Also the post-condition of *login* indicates the responsibility to create a *LoginRecord* instance. From the conceptual model and creator, *WbLS* is an appropriate candidate creator for the *LoginRecord* object. And by having the *WbLS* create *LoginRecord*, the *WbLS* can be associated with the *LoginRecord* object. Also the contract of *login* requires to modify *LoginDate* attribute of *LoginRecord*. This should be done by the *LoginRecord* itself, since it owns and maintains the attribute.
Collaboration diagram for **logout**

*WbLS* is the controller for handling this operation. The responsibility in the contract for **logout** indicates the need to find the user with userID. *WbLS* is a candidate for finding those from multiple objects. Also the post-condition of the contract for **logout** requires to modify *LogoutDate* and *LogoutTime* attributes of *LoginRecord*. This should be done by the *LoginRecord* itself.

**Figure 17.** A collaboration diagram for **logout** system operation

Collaboration diagram for **registerSystem**

*WbLS* is the controller for handling this operation. The post-conditions in the contract for **registerSystem** indicate the responsibility to create a Student and Teacher instances. From conceptual model, *WbLS* is a candidate creator for those objects. And by having the *WbLS* create Student and Teacher, the WbLS can be associated with the *Student* and *Teacher* objects over time by storing new instance in the collections of students or teachers.
**Collaboration diagram for registerSystem system operation**

WbLS is the controller for handling this operation. The responsibilities in the contract for registerCourse indicate the need to find the student and the course. From the conceptual model, Computer Science is a candidate for finding course from multiple objects, and WbLS is a candidate for finding student from multiple object. Also the post-condition of registerCourse indicates the responsibility to create a Registration instance. From the conceptual model and creator, WbLS is an appropriate candidate creator for the Registration object. And by having the WbLS create Registration, the WbLS can be associated with the Registration object over time. Date attribute of Registration is modified by the Registration itself.

**Collaboration diagram for registerCourse system operation**

CourseController is the controller added newly in conceptual model for handling this operation. The responsibility in the contract for selectCourseoff indicates the need to find the courseoffering. From the conceptual model, Course is a candidate for finding courseoffering from multiple
objects. Also the post-condition of selectCourseoff indicates the responsibility to create a Activity instance. From the conceptual model and creator, CourseController is an appropriate candidate creator for the Activity object. And by having the CourseController create Activity, the CourseController can be associated with the Activity object. IsAttempted attribute of Activity is set by the Activity itself.

**Figure 20.** A collaboration diagram for selectCourseoff system operation

**Collaboration diagram for selectLesson**

CourseoffController is the controller added newly in conceptual model for handling this operation. The responsibility in the contract for selectLesson indicates the need to find the lesson. From the conceptual model, CourseOffering is a candidate for finding lesson from multiple objects. Also the post-condition of selectLesson indicates the responsibility to create a Activity instance. From the conceptual model and creator, CourseoffController is an appropriate candidate creator for the Activity object. And by having the CourseoffController create Activity, the CourseoffController can be associated with the Activity object. IsAttempted attribute of Activity is set by the Activity itself.
Collaboration diagram for **selectLesson** system operation

**CourseoffController** is the controller for handling this operation. The post-condition of **endLesson** indicates the responsibility to modify a attribute of **Activity** class. This is done by the **Activity** itself.

Collaboration diagram for **endLesson**

**CourseoffController** is the controller added newly in conceptual model for handling this operation. The responsibility in the contract for **selectEmSheet** indicates the need to find the
examination sheet. From the conceptual model, *CourseOffering* is a candidate for finding the examination sheet from multiple object. Also the post-condition of *selectEmSheet* indicates the responsibility to create a *Activity* instance. From the conceptual model and creator, *CourseoffController* is an appropriate candidate creator for the *Activity* object. And by having the *CourseoffController* create *Activity*, the *CourseoffController* can be associated with the *Activity* object.

**Figure 23.** A collaboration diagram for *selectEmSheet* system operation

**Collaboration diagram for makeEmSolution**

*CourseoffController* is the controller for handling this operation. The post-condition of *makeEmSolution* indicates the responsibility to create a *Solution* instance. From the conceptual model and creator, *EmSheet* is an appropriate candidate creator for the *Solution* object. And by having the *EmSheet* create *Solution*, the *EmSheet* can be associated with the *Solution* object.
A WbLS system

Figure 24. A collaboration diagram for makeEmSolution system operation

Collaboration diagram for makeQuestion

CourseoffController is the controller for handling this operation. The post-condition of makeQuestion indicates the responsibility to create a Question instance. From the conceptual model and creator, Courseoffering is an appropriate candidate creator for the Question object. And by having the Courseoffering create Question, the Courseoffering can be associated with the Question object.

Figure 25. A collaboration diagram for makeQuestion system operation

Collaboration diagram for makeAnswer

CourseoffController is the controller for handling this operation. The post-condition of makeAnswer indicates the responsibility to create a Answer instance. From the conceptual model and
creator, Question is an appropriate candidate creator for the Answer object. And by having the Question create Answer, the Question can be associated with the Answer object.

**Figure 26.** A collaboration diagram for **makeAnswer** system operation

**Collaboration diagram for addCourse**

WbLS is the controller for handling this operation. The post-condition of addCourse indicates the responsibility to create a Course instance. From the conceptual model and creator, Computer Science is an appropriate candidate creator for the Course object. And by having the Computer Science create Course, the Computer Science can be associated with the Course object. The modification of CourseCount attribute of Computer Science class is done by the class itself.

**Figure 27.** A collaboration diagram for **addCourse** system operation

**Collaboration diagram for addLesson**

CourseoffController is the controller for handling this operation. The post-condition of addLesson indicates the responsibility to create a Lesson instance. From the conceptual model and
creator, *LectureNotes* is an appropriate candidate creator for the *Lesson* object. And by having the *LectureNotes* create *Lesson*, the *LectureNotes* can be associated with the *Lesson* object. The modification of *Count* attribute of *LectureNotes* class is done by the class itself.

![Collaboration diagram for addLesson operation](image)

**Figure 28.** A collaboration diagram for *addLesson* system operation

**Collaboration diagram for updateLesson**

*CourseoffController* is the controller for handling this operation. The post-conditions of *updateLesson* indicate the responsibility to modify the Content attribute a of *Lesson* instance. The modification of the attribute of *Lesson* class is done by the class itself.

![Collaboration diagram for updateLesson operation](image)

**Figure 29.** A collaboration diagram for *updateLesson* system operation

**Collaboration diagram for addEmSheet**
CourseoffController is the controller for handling this operation. The post-conditions of addEmSheet indicate the responsibility to create an EmSheet instance and an EmProblem instance. From the conceptual model and creator, Examination is an appropriate candidate creator for the EmSheet object. And by having the Examination create EmSheet, the Examination can be associated with the EmSheet object. Also by having the EmSheet create EmProblem, the EmSheet can be associated with the EmProblem object.

![Collaboration diagram for addEmSheet system operation](image)

**Figure 30. A collaboration diagram for addEmSheet system operation**

**Collaboration diagram for assessResult**

CourseoffController is the controller for handling this operation. The post-condition of assessResult indicates the responsibility to create an Assessment instance. From the conceptual model and creator, Solution is an appropriate candidate creator for the Assessment object. And by having the Solution create Assessment, the Solution can be associated with the Assessment object. The another post-condition of assessResult indicates the responsibility to set attributes of a Assessment class. The modification of Mark and Comment attributes of a Assessment class is carried out by the class itself.
Figure 31. A collaboration diagram for assessResult system operation

Collaboration diagram for addCourseoff

CourseController is the controller for handling this operation. The post-condition of addCourseoff indicates the responsibility to create a CourseOffering instance. From the conceptual model and creator, Course is an appropriate candidate creator for the CourseOffering object. And by having the Course create CourseOffering, the Course can be associated with the CourseOffering object.

Figure 32. A collaboration diagram for addCourseoff system operation
Collaboration diagram for *deleteCourseoff*

*CourseController* is the controller for handling this operation. The modification of *OfferingCount* attribute of *Course* class is carried out by the class itself.

![Collaboration diagram for deleteCourseoff](image)

**Figure 33.** A collaboration diagram for *deleteCourseoff* system operation

Collaboration diagram for *updateCourseoff*

*CourseController* is the controller for handling this operation. The modification of *startDate* attribute of *CourseOffering* class is carried out by the class itself.

![Collaboration diagram for updateCourseoff](image)

**Figure 34.** A collaboration diagram for *updateCourseoff* system operation

Collaboration diagram for *updateCourseActivity*

*OverallSequencing* is the controller for handling this operation. The responsibility in the contract for *updateCourseActivity* indicates the need to find the activity of the student for a course. From the conceptual model, *OverallSequencing* is a candidate for finding activity from multiple object. *IsCompleted* attribute of *Activity* is set by the *Activity* itself.
Collaboration diagram for **updateCourseActivity** system operation

**OverallSequencing** is the controller for handling this operation. The post-condition of **sequenceCourse** indicates the responsibility to create a **DeliverableCourse** instance. From the conceptual model and creator, **OverallSequencing** is an appropriate candidate creator for the **DeliverableCourse** multi object. And by having the **OverallSequencing** create **DeliverableCourse**, the **OverallSequencing** can be associated with the **DeliverableCourse** object.

Collaboration diagram for **sequenceCourse** system operation

**OverallSequencing** is the controller for handling this operation. The responsibility in the contract for **updateSeqCondition** indicates the need to find the activity of the student for a course.
From the conceptual model, *OverallSequencing* is a candidate for finding activity from multiple object. *Objective* attribute of *Activity* is set by the *Activity* itself.

![Collaboration diagram](image)

**Figure 37.** A collaboration diagram for `updateSeqCondition` system operation

**Collaboration diagram for** `sequenceLesson`

*OverallSequencing* is the controller for handling this operation. The post-condition of `sequenceLesson` indicates the responsibility to create a *DeliverableLesson* instance. From the conceptual model and creator, *OverallSequencing* is an appropriate candidate creator for the *DeliverableLesson* multi object. And by having the *OverallSequencing* create *DeliverableLesson*, the *OverallSequencing* can be associated with the *DeliverableLesson* object.

![Collaboration diagram](image)

**Figure 38.** A collaboration diagram for `sequenceLesson` system operation
Collaboration diagram for \textit{sequenceExam}

\textit{OverallSequencing} is the controller for handling this operation. The post-condition of \textit{sequenceExam} indicates the responsibility to create a \textit{DeliverableEmSheet} instance. From the conceptual model and creator, \textit{OverallSequencing} is an appropriate candidate creator for the \textit{DeliverableEmSheet} multi object. And by having the \textit{OverallSequencing} create \textit{DeliverableEmSheet}, the \textit{OverallSequencing} can be associated with the \textit{DeliverableEmSheet} object.

![Collaboration Diagram](image)

Figure 39. A collaboration diagram for \textit{sequenceExam} system operation

4.5.2 Design Class Diagrams

This section shows process creating \textit{design class diagrams} based on the identified software classes, the collaboration diagrams and the original conceptual model. \textit{Software classes} are classes representing the conceptual classes in the conceptual model and take a number of methods.

A design class diagram illustrates the information such as classes, associations, attributes, methods, types of attributes, navigability, and dependencies.

For example, when create the collaboration diagram for \textit{registerCourse} system operation, we can record methods of classes as follows.
For classes of objects participating in collaboration diagrams created earlier, we added methods as shown in the following Figure.
Course

Identifier : string
Catalog : string
Entry : string
Language : string
Description : string
Duration : string
Title : string
Prerequisite : string
Comment : string
Credit : int
KeyWord : string
StudentCount : int
OfferingCount : int
TargetObjective : string

CourseController

selectCourseoff(stID : string, coTitle : string, startDate : string)
addCourseoff(teID : string, coID : string, startDate : string)
deleteCourseoff(coID : string, offTitle : string)
updateCourseoff(coID : string, offTitle : string, startDate : string)

CourseoffController

selectLesson(stID : string, coTitle : string, startDate : string)
endLesson(stID : string, coTitle : string, startDate : string)
selectEmSheet(stID : string, coTitle : string, startDate : string)
makeEmSolution(stID : string, coTitle : string, startDate : string)
makeQuestion(stID : string, coTitle : string, startDate : string)
addLesson(teID : string, coID : string, startDate : string, duration : string)
updateLesson(coID : string, startDate : string, duration : string)
deleteLesson(coID : string, offTitle : string)
updateCourseoff(coID : string, startDate : string, duration : string)

Examination

makeEmSheet(stID : string)
setSheetCount()

ExamSheet

Identifier : string
Title : string
Max : int
Min : int
makeSolution(stID : string)
getSolution(stID : string)

Assessment

Mark : int
Comment : string
recordAssessment(mark : int, comment : string)

Solution

Description : string
CreatedDate : date
recordAssessment(mark : int, comment : string)

Answer

Description : string
Date : date

Question

ID : string
Question : string

makeAnswer()
In conceptual model created earlier, we assigned role at each end of an association. In a design class diagram, the role may be represented with a navigability. *Navigability* is a property of the role which indicates that it is possible to navigate unidirectionally across the association from objects of the source to the target class. Navigability is usually interpreted as attribute visibility from the source class to the target class.

We applied navigability to a conceptual model in Figure 13 and created design class diagram with association and navigability as shown in Figure 42.
Figure 42. Design Class Diagram with association and navigability
4.6 Implementation

4.6.1 Mapping a Design to Code

The end goal of an object-oriented development of a system is the creation of code in an object-oriented programming language. In the earlier design phase, we already created the information necessary in order to generate the code. Implementation in an object-oriented programming language requires writing source code class definitions and method definition. For this, we should basically define classes with methods and simple attributes, add reference attributes, and define methods from collaboration diagrams.

In this section, we show a simple example describing these definitions for a class created in design phase. For example, the definitions of methods and attributes of WbLS class by Java is described as follows.

```
public class WbLS {
    public login(string userID, string password);
    public logout(string userID);
    public registerSystem(string userID, string password, string type);
    public registerCourse(string stID, string coTitle);
    public addCourse(string teID, string coTitle, string comment);
    private setCount();
    private int StudentCount, TeacherCount;
    private ComputerScience cs;
    private LoginRecord log;
    private Person user;
    private Vector person;
    private Vector logRec;
}
```

**Figure 43. WbLS class in Java**
Based on these definitions, we can describe the process for login collaboration diagram as Java code (Figure 44).

```java
public void WbLS::registerCourse(string stID, string coTitle)
{
    co = cs.getCourse(coTitle);
    st = Student.elementAt(stID);
    rg = new Registration(st, co);
    rg.setDate();
    reg.addElement(rg);
}

public Course ComputerScience::getCourse(string coTitle)
{
    return Course.elementAt(coTitle);
}
```

Figure 44. Java code for registerCourse collaboration diagram

4.6.2 WbLS Conceptual Model Packages

As the conceptual model for development cycle two of our system is approaching an unwieldy size, it indicates the need to partition the model elements into smaller subsets. UML itself organized into packages. Each package contains some diagrams that mark up UML. Metaclasses are grouped together into packages depending on the degree of cohesion that they have with one another. Therefore, we grouped the elements into packages to support a higher-level view. WbLS domain conceptual package consists of seven packages.
A WbLS system

Figure 45. Domain concept package of WbLS

Core Package

Persons Package

Figure 47. Persons Package
Courses Package

Figure 48. Courses Package
Deliverables Package

Figure 49. Deliverables package
Sequencing Package

![Diagram of Sequencing Package]

**Figure 50.** Sequencing Package
Scheduling Package

![Scheduling Package Diagram]

**Figure 51.** Scheduling Package

Managements Package

![Managements Package Diagram]

**Figure 52.** Managements Package
5 Conclusions

Using UML, we have worked out a requirement specification and a design of the web-based learning system for teaching and learning Computer Science. We have concentrated on lessons, examination, homework and questions in learning activity component. Future work includes mapping the design specification of this project into code written in a programming language such as Java, and generalizing the models of this system to models of a prototype for a general subject. We will also study the problems of evolution and maintenance of distance learning systems. Also we will combine this design with relational or OO database such as SQL to store the information of persons and course material.

6 References


