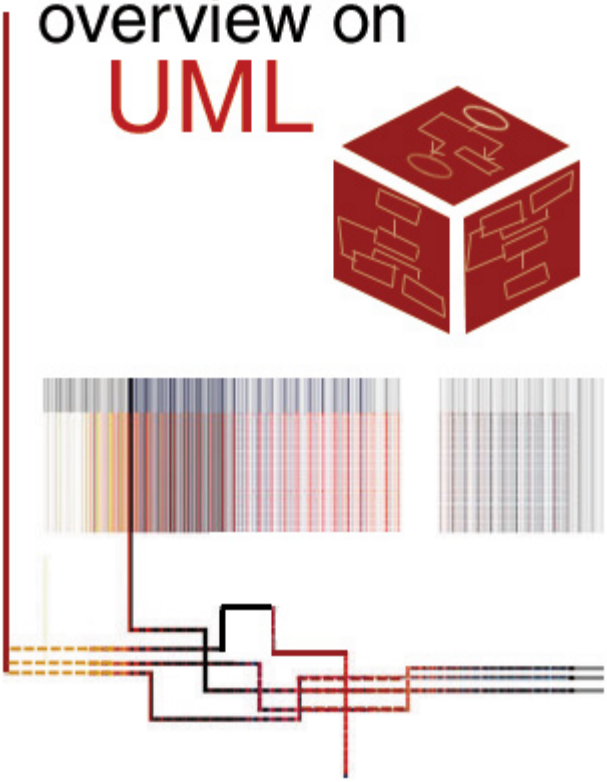
overview on UML



Sujatha Bhavikatti and Sunitha Sriram
Global TechPro, LLC
5659 Columbia Pike #200
Falls Church, Virginia 22041

Copyright © 2001-2002 Global TechPro. All rights reserved.

Visual Modeling: The Definition

Models are useful for understanding problems, communicating with everyone involved with the project (Customers, domain experts, analysts, designers etc.) modeling enterprises preparing documentation and designing programs and database. Modeling promotes better understanding of requirements, cleaner designs and more maintainable systems. They are abstractions that portray the essentials of a complex problem or structure by filtering out non-essential details, thus making the problem easier to understand.

Models help you to visualize a system as it is or as you want it to be. Models allow you to specify the structure or behavior of a system. They give you a template that guides you in constructing a system. Models document the decisions you have made.

Visual modeling provides a blueprint of a system. You wouldn't think of building a house without a complete set of architectural blueprints. The same is true with software.

Benefits of Visual Modeling

- **Capture Business Processes**

  When you create use cases, visual modeling allows you to capture business processes by defining the software system requirements from user's perspective.

- **Enhances Communications**

  Enhances communication between the Business domain (Capture business objects and logic) and Computer domain (Analyze and design the application). These teams must work together closely, but too often communication is strained by misunderstandings arising from differences in terminology.

Visual modeling has one communication standard, the UML, providing a smooth transition between the business domain and computer domain.

- **Manages Complexity**

  Visual modeling provides the capability to display modeling elements in many ways, so that they can be viewed at different levels of abstraction.

- **Defines architecture**

  Visual modeling provides the capability to capture the logical software architecture independent of the implementation language.

- **Enables reuse**

  With visual modeling one can reuse parts of a system or an application by creating components of one's design.

So how can you realize the benefits of modeling?

By using the UML (Unified Modeling Language).

Unified Modeling Language: The Definition

UML is the standard language for visualizing, specifying, constructing and documenting the artifacts of a software system. UML gives you a standard way write your system's blue prints. If you put UML to work at your organization, increased productivity, shorter development life cycles, and higher quality systems will become a reality.

History of UML

- Design methods popularized in 70's & 80's
- Technical community inundated with models, methodologies, notations by early 90's
- Standardization was needed, but no one was willing to champion the cause and make it successful; many were opposed to the idea
- OOPSLA '94 Grady Booch and James Rumbaugh announced the merging of their methods
- OOPSLA '95 revealed the first public description of the Unified Method, with Ivar Jacobson joining the duo
- The version 0.8 draft of Unified Method was released in October 1995
- During 1996, Grady Booch, Ivar Jacobson, and Jim Rumbaugh at Rational Software Corporation, with contributions from other leading methodologists, software vendors, and many users, worked on the new method, renaming it as the Unified Modeling Language (UML)
- UML 0.9 was released in June 1996
- In January '97, UML was proposed to OMG as a standard to facilitate the interchange of models; UML 1.1 adopted by OMG in Nov '97
- In June '98 UML 1.2 was released
- In Fall '98 UML 1.3 was released
- The UML provides the application modeling language for:
- Business process modeling with use cases.
- Class and object modeling.
- Component modeling.
- Distribution and deployment modeling.

Goals of the UML

- The primary goals in the design of the UML were as follows:
- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

Benefits of UML

UML does not guarantee project success but it does improve many things. For example, it significantly lowers the perpetual cost of training and retooling when changing between projects or organizations. It provides the opportunity for new integration between tools, processes, and domains. But most importantly, it enables developers to focus on delivering business value and provides them a paradigm to accomplish this. Some of the most important benefits are listed below.

- The OO advantage.
- Breaking down structures.
- Improving analysis.
- Management made simple.
- Linking teams.

UML Diagrams /Models

One of the goals of visual modeling is to understand the architecture of the application. The UML modeling diagrams will give the development team a full understanding of the application architecture. The choice of what models and diagrams one creates has a profound influence upon how a problem is attacked and how a corresponding solution is shaped. The tool used for designing the model is **Rational Rose**.

In terms of the views of a model, the UML defines the following graphical diagrams:

- Behavior diagrams:
  - Activity diagram
  - Interaction diagrams:
    - Sequence diagram
    - Collaboration diagram
- Use case diagram
- Class diagram
- Database diagram
- Implementation diagrams:
  - Component diagram
  - Deployment diagram

Activity diagram

One typically created earlier in the software life cycle representation, represents the dynamics of the system. Activity diagrams are the flow charts that show the flow of control from activity to activity, where each activity represents an operation on some class in the system. Activity diagrams may also be used later in the life cycle to show the workflow of operation.

In activity diagrams, the activities are represented by an oval symbol and flow of activity is represented by an arrow.

Use Case Diagram

Use case diagrams are graphical representation of overview of behavior of the system. Use case diagram shows the relationship between the actors and use cases in the system. Technically the use case diagram helps in defining the boundary of the system. The most important role of a use case diagram is to communicate the systems functionality and behavior to the customer or end user.

- **Actors**

  Some one or something that must interact &/or exchange the information with the system. An actor is represented by the following symbol.

- **Use cases**

  A use case is a sequence of transactions performed by a system that yields a measurable result of value(s) for a particular actor. The collection of use cases specifies the ways of using the system. Use cases are created by examining the actors and defining what they can accomplish with the system. A use case is represented by the following symbol.

Use cases can be captured graphically in two ways:

- **Sequence Diagram**

A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects and class involved in the use case and sequence of messages exchanged between the objects needed to carry out the functionality of the use case.

In sequence diagrams, rectangles represent the object, vertical dashed lines underneath each object represent the time and the horizontal arrows represent messages between the participating objects.

- **Collaboration Diagram**

A collaboration diagram shows object interactions organized around the objects and their link to each other. Collaboration diagrams are good for understanding all of the effects on a given object and are good for procedural design. One can either draw a Sequence or a Collaboration diagram, or both, to show implementation details of a scenario.

Class Diagrams

Class Diagrams show the static structure of the domain abstractions (classes) of the system**.** It describes the types of objects in the system and the various kinds of relationships that exist among them:

- **Association**
  Is the relationship that describes a set of links between or among objects. It is represented in the diagram by a line linking the classes.

- **Aggregation**
  Is the containment relationship between objects and is represented in the diagram with a line and a diamond symbol at the end.

- **Inheritance**

  Provides the capability to create a hierarchy of class where common structure and behavior are shared among classes. It is represented in the diagram with a line and a triangle symbol at the end.

  It shows the attributes and operations of some or all the classes in the model and the constraints for the way the objects collaborate.

  They may be created in the use case view. They are attached to the respective use cases containing a view of the classes participating in the use case. In order to analyze and discover classes from the diagrams we have developed so far, we need to understand what classes and objects really are.

- **Object – Definition**

  An Object is a representation of an entity, either real world or conceptual. It is a concept or abstraction with well-defined boundaries and meaning for an application.

  Each object has three characteristics: state, behavior and identity.
  State of an object is defined by a set of properties called attributes, with the values of the properties plus the relationships it has with other objects. The state of an object changes over time.
  Behavior determines how an object responds to requests from other objects and its capabilities. Behavior is implemented by a set of operations for the object.

  Identity means that each object is unique even if its state is identical to that of another object.

- **Class – Definition**

  A class is a description of a group of objects with common properties (attributes), common behavior (operations), and common relationships to other objects and common semantics.

  A class is a template to create objects. Each object is an instance of a class. Further more the related classes may be grouped into packages and package relationships defined. This gives a higher-level view of the model.

Packages

A **package** is a named general-purpose mechanism for organizing model elements, including, for instance, classes, use cases, diagrams, and/or other packages, into groups.

Package diagrams provide a mechanism for dividing and grouping model elements (e.g., classes, use cases). In UML, a package is represented as a folder.

Data Diagrams

The UML can be used to describe the complete development of relational and object relational databases from business requirements through the physical data model. However, modeling of the physical data model must express a detailed description of the database.

Provides an accurate model of the information needs of the organization and it is independent of any data storage and access method so that objective decisions can be made

Component Diagrams

Component diagrams reveal the structure of the code itself and the dependencies among software components, such as source code files, binary code files, executable files, or dynamic link libraries (DLLs). Typically, each component is then documented in more detail in a use case or class static structure diagram.

Deployment Diagram

A deployment diagram is a diagram that shows the physical architecture of the software and hardware of the system. The Deployment diagram is used to model the configuration of run-time processing elements and the software components, processes, and objects that live on them. In the deployment diagram, one can model the physical nodes and the communication associations that exist between them. Each node can contain run-time component instances, indicating that the component lives or runs on the node.

Case Study

Course registration at the local university is currently done by hand. Students fill out forms that contain their course selections and return the forms to the registrar. Clerks then enter the selections into a database and a process is executed to create student schedules. The registration process takes from one to two weeks to complete.

The university decided to investigate the use of an online registration system. This system would be used by professors to indicate the courses they would teach, by students to select courses, and by the registrar to complete the registration process.

Course Registration System Problem Statement

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

The new on-line registration system will allow students to select four course offerings for the coming semester. In addition, each student will indicate two alternative choices in case a course offering becomes filled or canceled. No course offering will have more than ten students. No course offering will have fewer than three students. A course offering with fewer than three students will be canceled. Once the registration process is completed for a student, the registration system sends information to the billing system, so the student can be billed for the semester.

Professors must be able to access the on-line system to indicate which courses they will be teaching. They will also need to see which students signed up for their course offering.

For each semester, there is a period of time that students can change their schedules. Students must be able to access the on-line system during this time to add or drop courses. The billing system will credit all students for courses dropped during this period of time.

Definition of Actors

The following actors were defined for the problem:

Student - someone who is registered to take courses at the University.

Professor - someone who is licensed to teach at the University.

Registrar - someone who is responsible for the maintenance of the Registration System.

Billing System - external system that bills students each semester.

Definition of Use Cases

The following use cases were elaborated for each actor:

Student

Register for courses.

Professor

Select courses to teach.

Request course offering roster.

Registrar

Generate course catalogue.

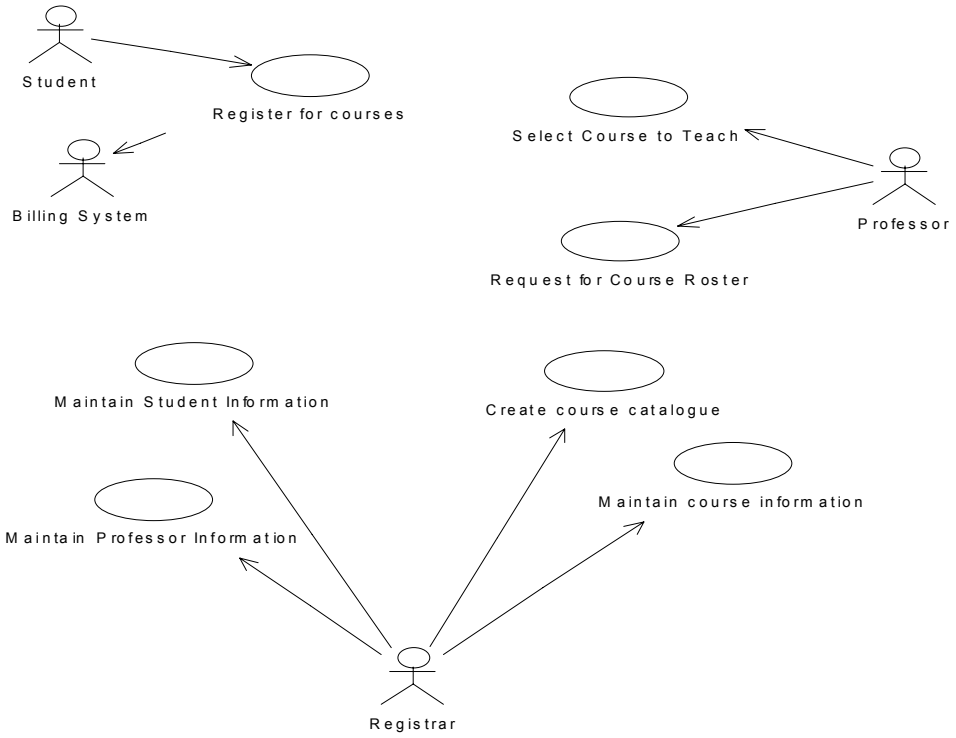
Maintain professor information.

Maintain student information.

Maintain curriculum.

Main Use Case Diagram for the Course Registration System

The use case diagram is contained within a class diagram in the use case view of the tool. Actors are shown as stickmen and use cases are shown as ovals. The use case diagram is shown below:



The Documentation field of the use case specification in the tool. A brief description of each use case follows:

- **Register for courses**

  The student starts the use case. It provides the capability to create, review, modify, and delete a course schedule for a specified semester. All pertinent billing information is sent to the Billing System.

- **Request class roster**

  The professor starts this use case. It provides the capability to request a printed list of all students assigned to a specified course offering.

- **Select courses to teach**

  The professor starts this use case. It provides the capability to select, review, modify, and delete a list of courses to teach for a specified semester.

- **Maintain professor information**

  The registrar starts this use case. It provides the capability to create, review, modify, and delete professor information.

- **Maintain student information**

  The registrar starts this use case. It provides the capability to create, review, modify, and delete student information.

- **Maintain curriculum**

  The registrar starts this use case. It provides the capability to create, review, modify, and delete a list of course offerings for a given semester.

- **Generate catalogue**

  The registrar starts this use case. It provides the capability to generate a catalogue containing a list of course offerings for a specified semester.

  During Inception, the flow of events (including any identified alternate flows) for the most important use cases is documented.

Flow of Events: Register for Courses Use Case

This use case begins when the student enters the student id number. The system verifies that the student id number is valid and prompts the student to select the current semester or a future semester. The student enters the desired semester. The system prompts the student to select the desired activity:

- Create a schedule.
- Review a schedule.
- Change a schedule:
  - Delete a course.
  - Add a course.

The student indicates that the activity is complete. The system will print the student schedule and notify the student that registration is complete. The system sends billing information for the student to the billing system for processing.

Alternate Flow

If an invalid id number is entered, the system will not allow access to the registration system.

If an attempt is made to create a schedule for a semester where a schedule already exists, the system will prompt for another choice to be made.

1. **Create a Schedule**

   The student enters 4 primary course offering numbers and 2 alternate course offering numbers. The student then submits the request for courses. The system then:

   - Checks that prerequisites are satisfied for the requested course.
   - Adds the student to the course offering if the course offering is open.

Alternate flow

If a primary course offering is not available, the system will substitute an alternate course offering.

2. **Review a Schedule**

   The student requests information on all course offerings in which the student is registered for a given semester. The system displays all courses for which the student is registered including course name, course number, and course offering number, days of the week, time, location, and number of credit hours.

3. **Change Schedule**

- **Delete a Course**

  The student indicates which course offerings to delete. The system checks that the final date for changes has not been exceeded. The system deletes the student from the course offering. The system notifies the student that the request has been processed.

- **Add a Course**

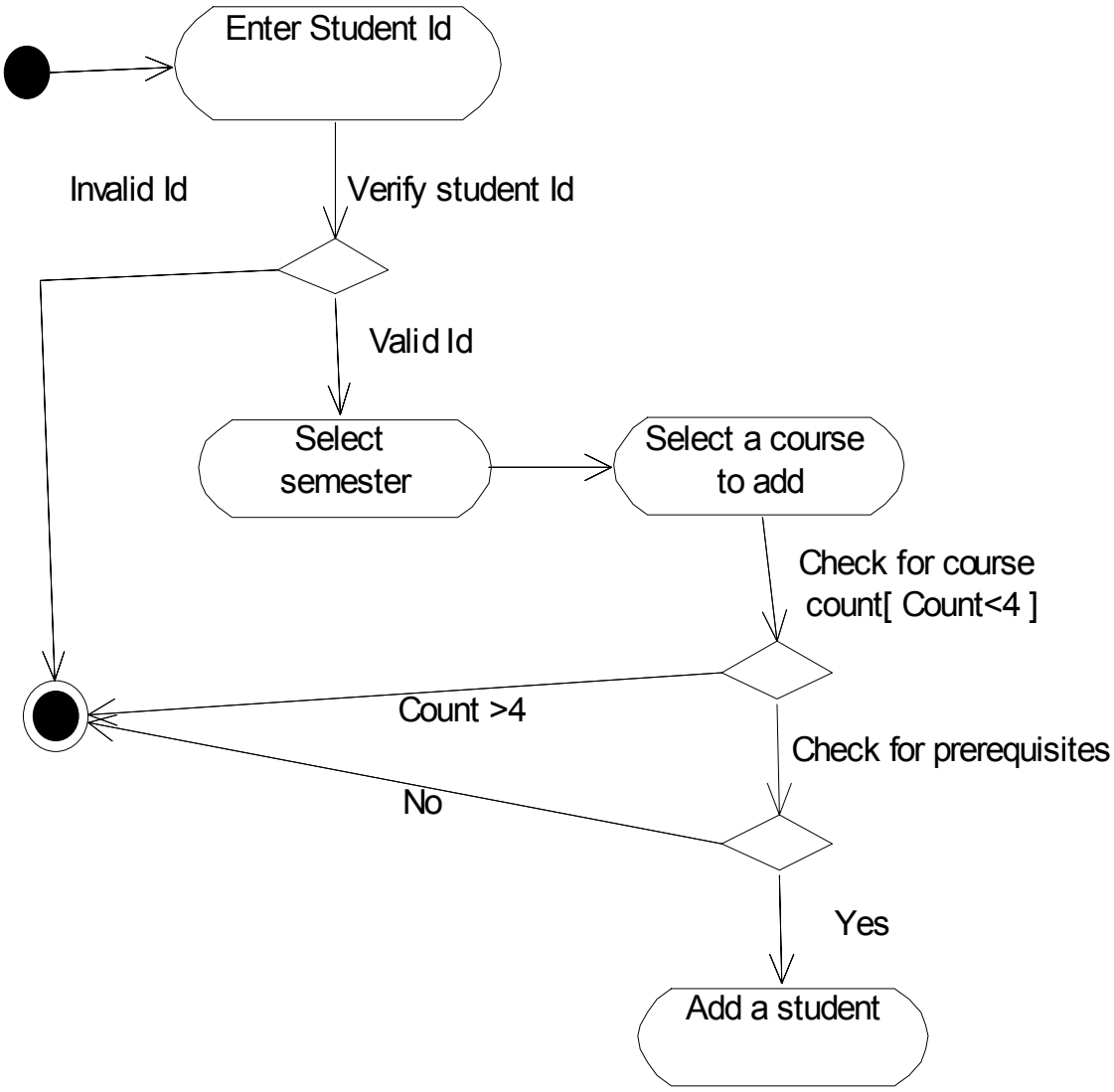
  The student indicates which course offerings to add. The system checks that the final date for changes has not been exceeded. The system then:

  Verifies that the maximum course load for the student has not been exceeded.

  Checks that prerequisites are satisfied for the requested course.

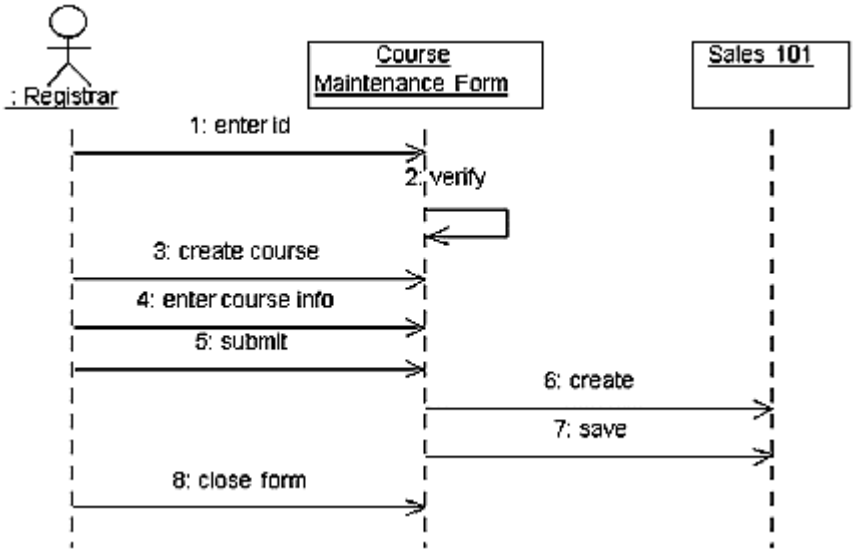
  Adds the student to the course offering if the course offering is open.

Activity diagram for add a course



Sequence Diagram for the Add A Course Use Case

The Sequence Diagram for the Add a Course scenario is shown in the figure below.



Creating "Real World" or "Business" Classes

The following packages and classes have been created for the registration system:

1. **People**

   StudentInfo - Information about the student actor needed by the registration system (for example, name, address, phone, idNumber, major, gradDate).

   ProfessorInfo - Information about the professor actor needed by the registration system (for example, name, address, phone, idNumber, tenure Status).

Class diagrams are created to graphically depict the packages and classes in the model. The Main class diagram typically contains only packages. Each package contains its own class diagrams. The Main class diagram for a package contains the public classes of the package (classes that communicate with classes in other packages). Other class diagrams are created as needed.

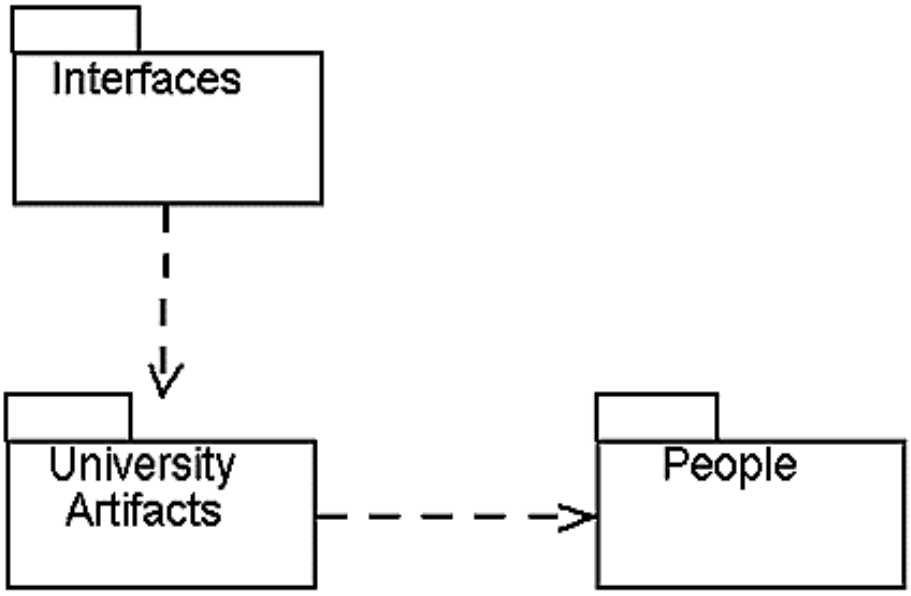
Use cases and **scenarios** are examined to determine the relationships needed by the system. Relationships between classes are created and displayed on selected class diagrams.

Attributes (structure) and operations (behavior) are added to the classes to carry out the functionality specified in the use cases.

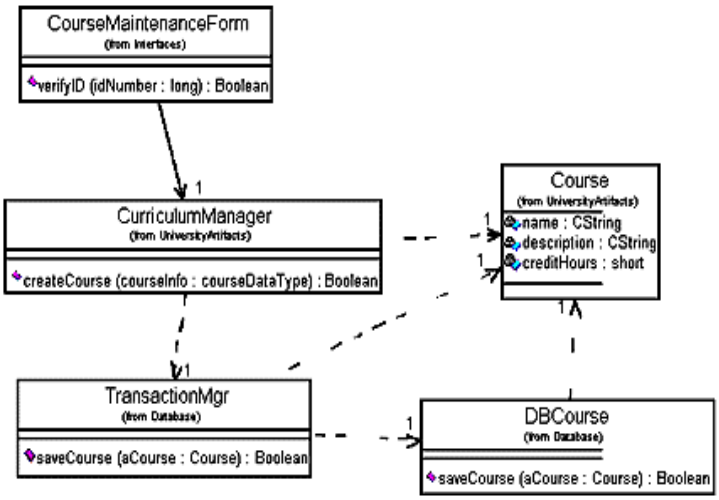
Sequence diagrams are updated to show the allocation of objects to classes and the replacement of messages with operations.

Some class diagrams for the Registration System are shown in the following figures.

Main Class Diagram

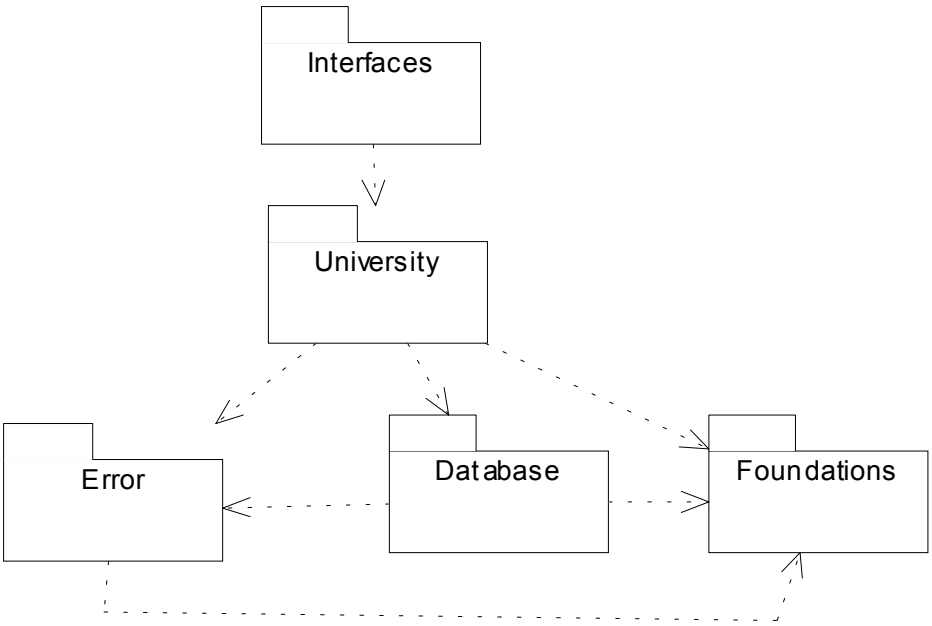


Main Class Diagram for the Add A Course

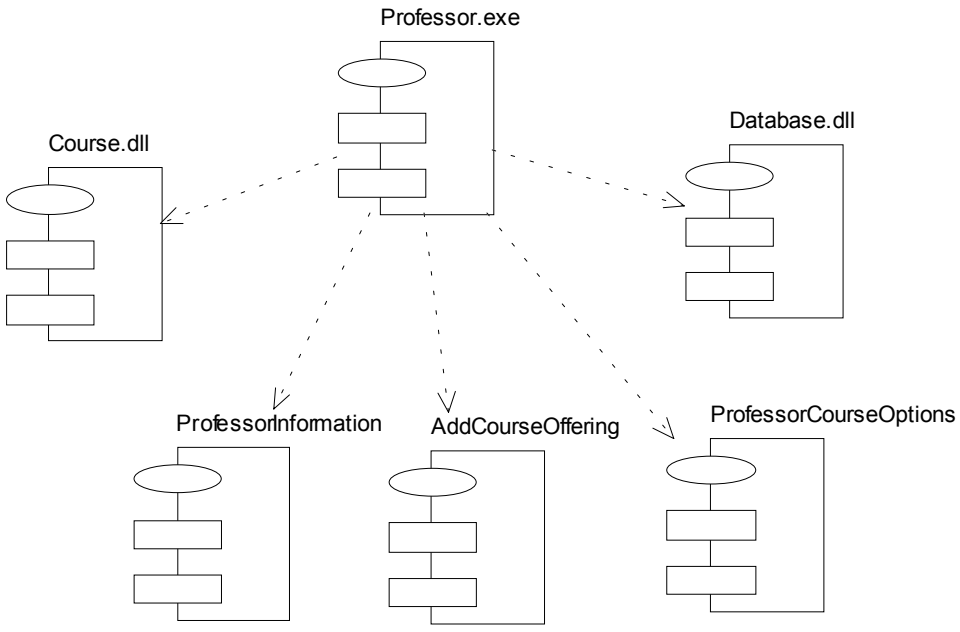


Main Component Diagram for the Course Registration System

The component view of architecture takes into account-derived requirements related to ease of development, software management, reuse and constraints imposed by programming languages and development tools.

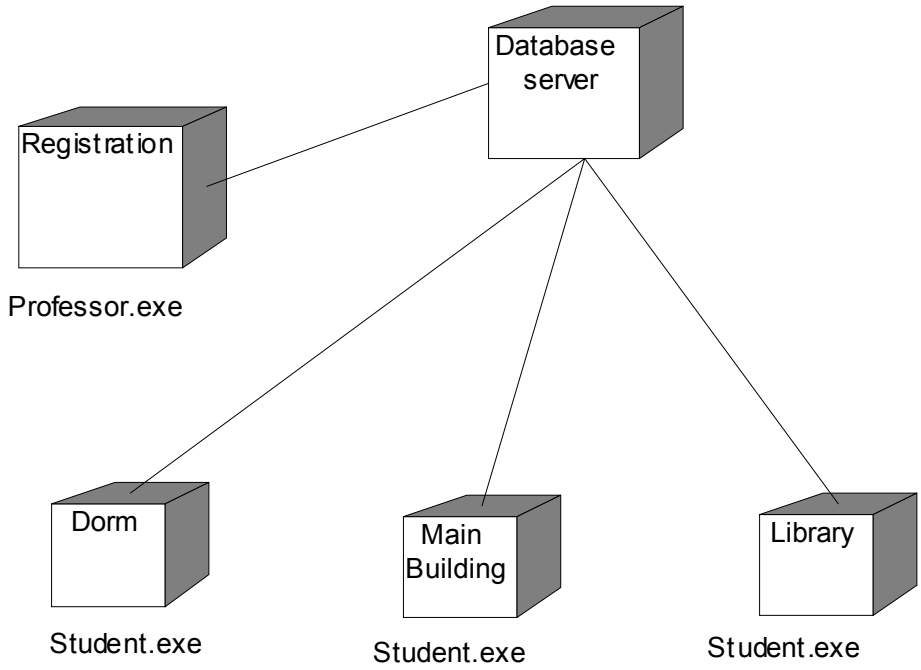


Component diagram for the Professor Executable (Professor.exe)



Deployment Diagram for the Course registration system

This view of architecture involves mapping software to processing nodes. It shows the configuration of run time processing elements and the software processes living in them.



Conclusion

This paper aims to present an overview of visual modeling with UML. It provides a brief understanding of OOPS and UML concepts. Models presented in this paper aims to give a more precise understanding of emerging software development techniques and concepts which can be used as the basis rigorous software development.

In the case of the course registration system, developing a model and implementing it would result in a paperless system.

Bibliography

1. Visual Modeling with rational rose and UML By Terry Quatrani
2. www.rational.com
3. www.raba.com
4. www.csai.unipa.it