

FINAL YEAR PROJECT REPORT ON STUDY OF RATIONAL ROSE

Submitted in Partial Fulfillment for the Requirement of
Degree of Bachelor of Engineering in Computer
Technology

By:

Amarendra Kumar (2k1/COE/06)
Anuj Dangri (2k1/COE/12)
Ashish Kumar Saini (2k1/COE/14)
Shikha Kochhar (2k1/COE/52)

Under the guidance of

Dr. Goldie Gabrani
Assistant Professor
Computer Engineering Department
Delhi College of Engineering

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI, DELHI-110042
2001- 2005

CONTENTS

Certificate

Acknowledgement

Abstract

List of Tables

List of Figures

1.	Welcome to Rational Suite		1
	1.1	Benefits of Using Rational Suite	1
	1.2	Rational Software Best Practices	2
2.	Rational Suite Tools		3
	2.1	Rational Requisite Pro	3
	2.2	Rational ClearQuest	3
	2.3	Rational ClearCase LT	3
	2.4	Rational SoDA	4
	2.5	Rational TestManager	4
	2.6	Rational ProjectConsole	4
	2.7	Rational ClearCase	5
	2.8	Rational XDE	5
	2.9	Rational Developer Network	5
	2.10	Rational Professional Services	5
	2.11	Rational Purify Plus	5
	2.11.1	Rational Purify	
	2.11.2	Rational Quantify	
	2.11.3	Rational PureCoverage	
	2.12	Rational Project Console	6
	2.13	Rational Robot	6
	2.14	Rational Rose	7
	2.15	Unified Change Management (UCM)	7
	2.16	Rational Unified Process (RUP)	7
	2.16.1	Along Time	
	2.16.2	Along Process	
3.	RATIONAL ROSE Introduction		10
	3.1	Other CASE TOOLS	
4.	History of Rational Rose		12
5.	Rational Rose Features		13
6.	Visual Modeling		14

7.	UML (Unified Modeling Language)	15
	7.1 History of UML	
	7.2 UML Description	
	7.3 Goals of UML	
	7.4 UML STRUCTURE	
8.	Modeling with Rational Rose	18
	8.1 Application Window	
	8.1.1 Title Bar	
	8.1.2 Menu Bar	
	8.1.3 Toolbar	
	8.1.4 Toolbox	
	8.2 Browser window	
	8.3 Documentation window	
	8.4 Log window	
	8.5 Diagram window	
	8.6 Overview window	
	8.7 Specification window	
9.	Introduction to Diagrams	24
	9.1 Deleting Model Elements	
	9.2 Correlations	
	9.3 Laying Out a Diagram	
10.	Class / Object diagram	29
	10.1 Overview	
	10.2 Association	
	10.3 Aggregation	
	10.4 Generalization	
	10.5 Types of Classes	
	10.5.1 Entity Class	
	10.5.2 Boundary Class	
	10.5.3 Control Class	
	10.6 Package	
	10.7 Class Diagram Toolbox	
	10.8 Class Specification	
	10.9 Cardinality	
	10.10 Persistence	
	10.11 Abstract	
	10.12 Operations Tab	
	10.13 Attributes Tab	
	10.14 Nested Tab	
	10.15 Containment	

11. Use Case Diagrams	37
11.1 Actors	
11.2 Use Case	
11.3 Flow of Events	
11.4 Relationships	
11.5 Association	
11.6 Dependency	
11.6.1 Extend Stereotype	
11.6.2 Include Stereotype	
11.6.3 Refine Stereotype	
11.7 Generalization	
11.8 Use Case Diagram Toolbox	
11.9 Use Case Specification	
11.10 Specification Content	
11.11 Name	
11.12 Rank	
11.13 Abstract	
12. State Machine Diagrams	43
12.1 State Machine Specification	
12.2 State chart diagram	
12.2.1 Creating a Statechart Diagram	
12.3 Activity diagram	
12.3.1 Using Activity Diagrams	
12.3.2 Understanding Workflows	
12.3.3 Creating an Activity Diagram	
12.3.4 Activities	
12.3.5 Swim lanes	
12.3.6 Objects	
12.3.7 Object Flow	
12.3.8 States	
12.3.9 Transitions	
12.3.10 Decisions	
12.3.11 Synchronizations	
12.3.12 State and Activity Actions	
12.3.13 Guard Condition	
13. Interaction Diagrams	49
13.1 Creating and Displaying an Interaction Diagram	
13.2 Collaboration Diagrams	
13.2.1 Collaboration Diagram Toolbox	
13.3 Sequence Diagrams	

	13.3.1 Sequence Diagram Toolbox		
	13.3.2 Object		
	13.3.3 Multiple Objects		
	13.3.4 Messages		
	13.3.5 Links		
	13.3.6 Sequence Numbering		
	13.3.7 Scripts		
	13.3.8 Focus of Control		
14.	Component Diagrams	54	
	14.1 Overview		
	14.2 Creating and Displaying a Component Diagram		
	14.3 Component Diagram Toolbox		
15.	Deployment Diagrams	55	
	15.1 Overview		
	15.2 Creating and Displaying a Deployment Diagram		
	15.3 Deployment Diagram Toolbox		
	15.4 Processor Specification		
	15.5 Processor		
	15.6 Priority		
	15.7 Characteristics		
	15.8 Scheduling		
16.	Case Study: Student Registration System	57	
	16.1 Problem Statement		58
	16.1.1 Case Study Background		
	16.1.2 Problem Statement		
	16.1.3 Project Summary		
	16.2 The Inception Phase	59	
	16.2.1 Business Goals and Needs		
	16.2.2 Definition of Actors		
	16.2.3 Use Cases		
	16.2.4 Use Case Diagram in Rational Rose		
	16.2.5 Flow of Events: Register for Courses Use Case		
	16.2.5.1 Create a Schedule		
	16.2.5.2 Review a Schedule		
	16.2.5.3 Change Schedule – Delete a Course		
	16.2.5.4 Change Schedule – Add a Course		
	16.2.6 Flow of Events: Select Courses to Teach		

	16.2.6.1 Brief Description		
	16.2.6.2 Basic Flow		
	16.2.6.3 Alternate Flow		
	16.2.7 Activity Diagram		
16.3	The Elaboration Phase	67	
	16.3.1 Development of Scenarios		
	16.3.2 Creating "Real World" or "Business" Classes		
	16.3.3 Software Architecture		
	16.3.4 Iteration Planning		
16.4	The Construction Phase	75	
	16.4.1 Construction Activities		
	16.4.2 Building Iteration		
	16.4.3 Add a Course		
	16.4.4 State Chart Diagram		
	16.4.5 Main Component Diagram		
16.5	The Transition Phase	80	
	16.5.1 Deployment View		
17.	Case Study: Online Auction System	81	
	17.1 Problem Statement		82
	17.2 Real Time Auction System Glossary	83	
	17.3 System Requirement Specifications	85	
	17.4 Flow of Events	106	
	17.5 USE CASE Diagram		107
	17.6 Class Diagram	108	
	17.7 Sequence Diagram	109	
	17.7.1 Placing a BID:		
	17.7.2 Join:		
	17.7.3 Listing an Item:		
	17.7.4 Login by User:		
	17.7.5 View Profile:		
17.8	Collaboration Diagram	112	
	17.8.1 Placing Bid:		
	17.8.2 Join:		
	17.8.3 Listing Item		
	17.8.4 View Profile:		
	17.8.5 Login by USER:		
	17.9 Activity Diagram	115	
	17.10 State Transition Diagram		116
	17.11 Deployment Diagram	117	
18.	Conclusion	118	
19.	References	119	

CERTIFICATE

This is to certify that the dissertation entitled “Study Of Rational Rose” being submitted by Amarendra Kumar (2k1/COE/06), Anuj Dangri (2k1/COE/12), Ashish Kumar Saini (2k1/COE/14), Shikha Kochhar (2k1/COE/52) towards the partial fulfillment of the requirement for the award of the degree of BACHELOR OF ENGINEERING in Computer Engineering at Delhi College Of Engineering, Delhi, is a bona fide record of their work carried out under my supervision.

Further it is also certified that the matter and results in this dissertation are original and have not been submitted for any degree or diploma in any other college to best of my knowledge.

Dated: 24/5/2005

Dr. Goldie Gabrani
Assistant Professor
Department of Computer Engineering
Delhi College of Engineering, Delhi.

ACKNOWLEDGEMENT

We wish to express our heartily and sincere gratitude and indebtedness to Assistant Professor Dr. Goldie Gabrani, Computer Engineering Department, Delhi College of Engineering, Delhi for her invaluable guidance and wholehearted cooperation. She has been a major source of inspiration throughout the project as she not only guided us through the project but encouraged me to solve the problems that arose during this project. Her extreme knowledge in this area helped us to complete this study project on Rational Rose. It was a great experience working under her.

Our heartily thanks to all our professors for their expertise and all rounded personality they have imparted to us.

Amarendra Kumar
Anuj Dangri
Ashish Kumar Saini
Shikha Kochhar

ABSTRACT

Rational Rose is a CASE Tool, a category of software that provides a development environment for programming teams. It helps to automate, manage and simplify the Software development process. These can include tools for: Summarizing initial requirements, Developing flow diagrams, Scheduling development tasks, preparing documentation, controlling software versions and developing program code.

Rational Rose is the visual modeling software solution that lets you create, analyze, design, view, modify, and manipulate components. You can graphically depict an overview of the behavior of your system with a use-case diagram. Rational Rose provides the collaboration diagram as an alternative to a use-case diagram. It shows object interactions organized around objects and their links to one another. The statechart diagram provides additional analysis techniques for classes with significant dynamic behavior. Activity diagrams provide a way to model a class operation or the workflow of a business process. The logical architecture is captured in class diagrams that contain the classes and relationships that represent the key abstractions of the system under development. The component architecture is captured in component diagrams that focus on the actual software module organization within the development environment. The deployment architecture is captured in deployment diagrams that map software to processing nodes—showing the configuration of run-time processing elements and their software processes.

The thesis starts with an introduction to Rational Suite Components and how the use of rational suite makes the development process simplified. Then a brief description of Visual modeling is given followed by UML description and how a

system is depicted in UML notations. Then a brief about Rational Rose is provided with its features. How to start working in Rational Rose is summarized via the Interface area of Rational Rose. Namely interface is divided into: Application window, Browser window, Documentation window, Diagram window, Overview window, Specification window and Log window.

Then various diagrams which are supported by Rational Rose are explained, namely: Class diagram, Object diagram, Use-case diagram, Collaboration diagram, Sequence diagram, Component diagram, Statechart diagram, Deployment diagram and Activity diagram.

After this much of theoretical introduction two case studies are done, namely: Student Registration System and Online Auction System. For both case studies all diagram windows are created and the overall complex structure for problems is simplified by means of object orientation.

List of Tables

Table 1: Rational Software Practices

Table 2: RUP Features

Table 3: Export Options

Table 4: Cardinality Options

Table 5: Physical Containment Options

Table 6: Scheduling Options

List of Figures

{Rational Rose Introduction}

- Figure 1: Time and Process Components of RUP
- Figure 2: UML Unification
- Figure 3: UML Views
- Figure 4: Application Window
- Figure 5: Application window toolbar
- Figure 6: Diagram Window
- Figure 7: Multiple Diagrams—Cascade Windows
- Figure 8: Multiple Diagrams—Tiled Windows
- Figure 9: Diagram Layout
- Figure 10: Association example
- Figure 11: Aggregation relationship
- Figure 11: Generalization in classes
- Figure 12: Control, Entity and Boundary Class
- Figure 13: Packages
- Figure 14: Class Toolbox
- Figure 15: Notation for an Actor
- Figure 16: Notation for Use Case
- Figure 17: Dependency Display
- Figure 18: Use Case Toolbox
- Figure 19: UML Notation of a state
- Figure 20: Automatic Transmission Example
- Figure 21: UML Notation of activity
- Figure 22: Activity diagram
- Figure 23: Object Flow
- Figure 24: Collaboration Example
- Figure 25: Collaboration Toolbox
- Figure 26: Sequence Diagram
- Figure 27: Sequence Toolbox
- Figure 28: Component Diagram
- Figure 29: Component Toolbox
- Figure 30: Deployment Example
- Figure 31: Deployment Toolbox

{User Registration System}

- Figure 32: Use Case Diagram
- Figure 33: Use Case Dependency

Figure 34: Activity Diagram: Create Catalogue
Figure 35: Sequence Diagram for the Add a Course Scenario
Figure 36: Main Class Diagram
Figure 37: Main Class Diagram for the People Package
Figure 38: Main Class Diagram for the University Artifacts Package
Figure 39: Course Reporting Class Diagram in the University Artifacts Package
Figure 40: Main Class Diagram for the Interfaces Package
Figure 41: Updated Sequence Diagram
Figure 42: Class Diagram with Class types
Figure 43: Sequence Diagram for Registrar
Figure 44: Collaboration Diagram for Registrar
Figure 45: Main Class Diagram
Figure 46: Updated Sequence Diagram
Figure 47: Updated Sequence Diagram
Figure 48: Class Diagram "Add a Course"
Figure 49: Realization Diagram
Figure 50: Sequence Diagram for Add a course offering
Figure 51: State Chart Diagram for Course Offering States
Figure 52: Main Component Diagram
Figure 53: University Main Component Diagram
Figure 54: Main Deployment Diagram

{Online Auction System}

Figure 55: Use Case Diagram
Figure 56: Class Diagram
Figure 57: Sequence Diagram for placing a bid
Figure 58: Sequence Diagram for Join
Figure 59: Sequence Diagram for listing an item
Figure 60: Sequence Diagram for Login
Figure 61: Sequence Diagram for Viewing Profile
Figure 62: Collaboration Diagram for Placing bid
Figure 63: Collaboration Diagram for Join
Figure 64: Collaboration Diagram for listing an item
Figure 65: Collaboration Diagram for Viewing Profile
Figure 66: Collaboration Diagram for Login
Figure 67: Activity Diagram for Auction System
Figure 68: State Transition Diagram for Auction System
Figure 69: Deployment Diagram for Auction System

1. Rational Suite Introduction

In 1990s, a number of large sized projects failed called Software Runaways and resulted into so called 'Software Crisis'. Statistics show that only 2% of the projects were used as they were delivered, 3% of the projects used after modifications, 47% of the projects were never delivered, 19% of the software projects rejected or reworked and 29% was not even delivered. The problem increased because of increased dependence of business on software and lack of systematic approach to build the software.

Some of the examples are:

- In June 1996, Anane 5 launcher broke up and exploded after 40 seconds of take off at an altitude of less than 4 Kilometers. The total loss was \$500 million. Problem found was due to overflow in conversion from a 64bit floating point number to 16 bit signed integer.
- Therac-25 a radiation therapy and X-ray machine killed several patients due to malfunctioning of arrow keys which were not programmed properly by designers. As a result high dose of radiations was given to patients.
- Ministry of agriculture in UK alone had to undergo a loss of 12 million pounds because of software errors.
- Even the launch of space shuttle Columbia was delayed by three years thus costing millions of dollars.

The main Project Failure factors were:

- Projects finish late (or not at all).
- Overrun their Budget
- Unable to meet Requirements.
- Unsatisfied users and unreliable Performance.
- Unclear communication among team members.
- Serious design flaws are uncovered late in development.

1.1 Benefits of Using Rational Suite

Rational Software helps organizations overcome these challenges and develop Software more successfully by offering:

- Software engineering best practices.
- Integrated tools that automate these best practices.
- Professional services that accelerate adoption and implementation of these best practices and tools.
-

Rational puts these best practices to work by offering tools that:

- Unify teams and enhance communication.
- Optimize individual productivity.
- Simplify adoption with common installation, licensing, and user support plans.

Rational Suite editions are customized with sets of tools best suited for each member of your team.

1.2 Rational Software Best Practices

Best practice	Project lifecycle tasks	Benefits
Develop software iteratively.	Analyze, design, and implement incremental subsets of the system.	<ul style="list-style-type: none"> ▪ Mitigate risks earlier. ▪ Achieve higher quality. ▪ Increase reusability of project assets.
Manage requirements.	Find, document, organize, and track changing requirements.	<ul style="list-style-type: none"> ▪ Mitigate risks associated with change. ▪ Ensure products meet their stated goals.
Use component architectures.	<p>Design the structural elements of a system, how those elements behave, and decide how they fit into progressively larger systems.</p> <p>Identify and decide which subsystem components to develop, reuse, and buy.</p>	<ul style="list-style-type: none"> ▪ Improve product reliability, maintainability, and extensibility.
Model visually.	Create graphical designs of system architectures.	<ul style="list-style-type: none"> ▪ Manage design complexity. ▪ Improve understanding of complex systems. ▪ Explore and compare design alternatives at a low cost. ▪ Form a basis for implementation.
Continuously verify quality.	Test in each iteration.	<ul style="list-style-type: none"> ▪ Verify that the system delivers required functionality, reliability, and performs under load. ▪ Discovers problems with requirements, designs, and implementations early.
Manage change.	Establish repeatable development process.	<ul style="list-style-type: none"> ▪ Improve team communication. ▪ Control risk. ▪ Improve resource allocation. ▪ Enable parallel development.

Table 1: Rational Software Practices

2. Rational Suite Tools

2.1 Rational Requisite Pro

Helps you manage requirements more effectively by using a database to:

- Organize requirements.
- Prioritize requirements.
- Track requirements.

- Develop requirement documents.

RequisitePro includes the RequisiteWeb interface so users can view, create, and manage requirements from a Web browser. RequisitePro is available for Windows only. RequisiteWeb is available for Windows and UNIX.

2.2 Rational ClearQuest

Manages change activity associated with software development, including:

- Enhancement requests.
- Defects.
- Documentation modifications.

The ClearQuest Web interface allows Windows and UNIX users to perform basic ClearQuest operations from a Web browser, such as submitting and finding records, Creating or editing queries and reports, and creating shortcuts. ClearQuest MultiSite enables sharing of information across a geographically distributed team. ClearQuest is available for Windows and UNIX client workstations. For either platform, you must configure a Windows workstation as the administrator for the ClearQuest repository.

2.3 Rational ClearCase LT

Provides software configuration management and a built-in process to track changes to all software project assets, including:

- Requirements.
- Visual models.
- Code.
- Documentation.

It also provides a Web interface that lets users perform basic ClearCase LT operations. Rational ClearCase LT supports *Unified Change Management*, the rational process for managing change and controlling workflow.

ClearCase LT is available for Windows and UNIX. Rational ClearCase LT is a configuration management tool for small project teams.

Analysts use it to manage changes to requirements. ClearCase LT lets you associate requirements with:

- Specific releases.
- Code.
- Visual models.
- Test scripts.

Because ClearCase LT tracks changes to all your project files, team members can work in parallel and integrate their changes to the project baseline.

Rational ClearQuest is a change management tool used to track defects, new features, product enhancements, and other change requests. It helps you:

- Evaluate change requests.
- Determine their impact on the system.
- Validate the changes (when applicable).

To establish how change requests fit into the structure of features and main requirements, you can link requests to an existing or new project requirement in Rational RequisitePro.

2.4 Rational SoDA

It automatically generates project documents by extracting information from development artifacts, including files produced by Rational tools and source code. SoDA uses predefined or customized templates to format the information.

SoDA is integrated with Microsoft Word for Windows and Adobe FrameMaker for UNIX.

2.5 Rational TestManager

It helps you create realistic functional and multiuser tests to assess the performance and reliability of Web, multitier, and database applications.

TestManager tracks:

- How many tests have been planned, developed, and run.
- Which requirements have been tested?
- How many tests have passed and failed.

Using this data, TestManager helps you objectively assess project status and create reports to communicate these findings to project stakeholders.

TestManager is available for Windows client workstations and UNIX agents as long as a Windows workstation is configured as the administrator for the TestManager repository.

2.6 Rational ProjectConsole

It helps you track project metrics by automatically generating charts and gauges from data produced during software development. ProjectConsole is integrated with Microsoft Project so that you can create, display, and report on a centralized project plan. ProjectConsole organizes project artifacts on a central Web site so all team members can view them. ProjectConsole is available for Windows only.

2.7 Rational ClearCase

Rational ClearCase is the configuration management solution for large projects and teams. In addition to ClearCase LT features, it offers:

- Advanced build management.
- Distributed server support.
- Automatic data replication.

2.8 Rational XDE

Rational XDE is an extended development environment that provides visual design and development capabilities. Integrated with Microsoft Visual Studio .NET and IBM WebSphere, XDE gives developers a single user experience. It can also be used alone through the Rational supported Java platform Integrated Development Environment (IDE), based on the Eclipse platform.

2.9 Rational Developer Network

The Rational Developer Network (RDN) is an online community for all Rational customers. RDN provides:

- A forum for exchanging ideas and best practices with other software professionals.
- Web-based training.
- Articles, white papers, *Getting Started* programs, and other content.

2.10 Rational Professional Services

Rational also offers:

- Consulting and mentoring.
- Packaged services, such as “Quick starts,” deployment services, and workshops.
- Customized services and projects.

2.11 Rational PURIFYPLUS

Rational PurifyPlus brings together three essential tools that help you develop high-quality applications more efficiently:

2.11.1 Rational Purify: An automatic error detection tool for finding runtime errors and memory leaks in every component of your program.

2.11.2 Rational Quantify: A performance analysis tool for resolving performance bottlenecks. Rational Quantify helps testers assess quality by: Pinpointing places in the application where the code is running

inefficiently and helping to identify the cause so developers can improve system performance.

2.11.3 Rational PureCoverage: A code coverage tool for making sure your code is thoroughly tested before you release it.

Benefits of Using Rational PurifyPlus are:

- **Find memory errors early:** Use Purify as you code to pinpoint hard-to-find bugs. Memory errors don't always show up right away, but they're the ones that will make your program crash someday.
- **Prevent performance bottlenecks:** Whenever you write new code or modify existing code, use Quantify to catch any incremental performance losses before they turn into bottlenecks. Quantify gives you the information you need to write more efficient code. It can turn everyone on your team into a performance engineer.
- **Improve code coverage:** Use PureCoverage to make sure you're exercising all your code during pre-checkin testing. For C/C++ code, you can run PureCoverage from within Purify—just click Coverage, error, and leak data in Purify's Run Program dialog.
- **Analyze code structure:** A common reason for writing new code is to improve the performance of a program. But how can you effectively improve the performance of code that might have been developed over several years by many different people? Use Quantify not only to find performance bottlenecks, but also to learn more about how your code is structured. It will help you to make effective performance improvements.

2.12 Rational Project Console

It helps you track project metrics by automatically generating charts and gauges from data produced during software development. ProjectConsole organizes project artifacts on a central Web site so all team members can view them. It is integrated with Microsoft Project in Windows.

2.13 Rational Robot

It determines whether the system meets requirements by testing how it responds to a user-driven scenario.

With Robot, you can:

- Record a test.
- Insert verification points to monitor expected results.
- Replay tests as often as needed.
- Run tests.

- View test results and details of test failures such as which test was running, what type of failure occurred, where the failure occurred, and which verification point failed.

2.14 Rational Rose

Rational Rose provides support for two essential elements of modern software engineering: component-based development and controlled iterative development. While these concepts are conceptually independent, their usage in combination is both natural and beneficial. Rational Rose's model-diagram architecture facilitates use of the Unified Modeling Language (UML), Component Object Modeling (COM), Object Modeling Technique (OMT), and Booch '93 method for visual modeling. Using semantic information it ensures correctness by construction and maintaining consistency.

2.15 Unified Change Management (UCM)

Unified Change Management (UCM) is the built-in Rational process that helps you use ClearCase LT with ClearQuest to:

- Define how to manage changing requirements, design models, documentation, components, and source code.
- Link the activities used to plan and track software development with the artifacts that are changing throughout development.

UCM is available for use with ClearCase LT on Windows only.

UCM Example

1. A project leader uses ClearQuest to assign an *activity* — change request — to a developer.
2. ClearCase LT with UCM maintains a *change set* — a list of changed artifacts — for the activity.
3. The developer works on the activity, creating or modifying artifacts associated with it.
4. The developer *delivers* the work to the *integration stream* — a project-wide, shared workspace.
5. Within this stream, team members can build and test the latest versions of the project's shared elements.
6. After, the project leader may decide to create a new *baseline* — a stable configuration of a project's components that becomes the basis for future development work.

2.16 Rational Unified Process (RUP)

It helps you create and implement a development process so your team can work more efficiently and communicate more effectively.

You can configure the RUP to:

- Use only the process components you need for each stage of a project.
- Develop your own process components.
- Exchange best practices with peers and industry leaders.

The RUP is available for Windows and UNIX.

Sustained delivery of high-quality software requires cohesive teamwork and a common understanding of development tasks. That is why implementing a predictable, repeatable process like the RUP is important to your success.

RUP Features	RUP Benefits
A library of processes and tools.	Select and deliver the best process for your projects.
A customizable Web interface so that you can design the process configuration.	Establish organizational process standards.
Tool Mentors provide instructions for using Rational tools. Extended Help (Windows only) from all Rational tools links to relevant topics in the RUP. You can also customize it by adding your own content.	Learn how to use Rational tools in your process.

Table 2: RUP Features

RUP is structured along two dimensions

- Time: division of the life cycle into phases and iterations.
- Process components: production of a specific set of artifacts with well-defined activities.

2.16.1 Along Time:

- Inception—Define scope of project(project vision).
- Elaboration—Plan project, specify features, baseline architecture, required resources.
- Construction—Build and test product.
- Transition—Deliver product to user community (manufacturing, delivering and training).

2.16.2 Along Process Components:

- Business Modeling – the identification of desired system capabilities and user needs
- Requirements – a narration of the system vision along with the a set of function and non-function requirements
- Analysis and Design- a description of how the system will be realized in the implementation phase.

- Implementation- the production of the code that will result in an executable system.
- Test-the verification of the entire system.
- Deployment-the delivery of the system and user training to the customer.

Each activity of the process component dimension typically is applied to the each phase of the time based dimension.

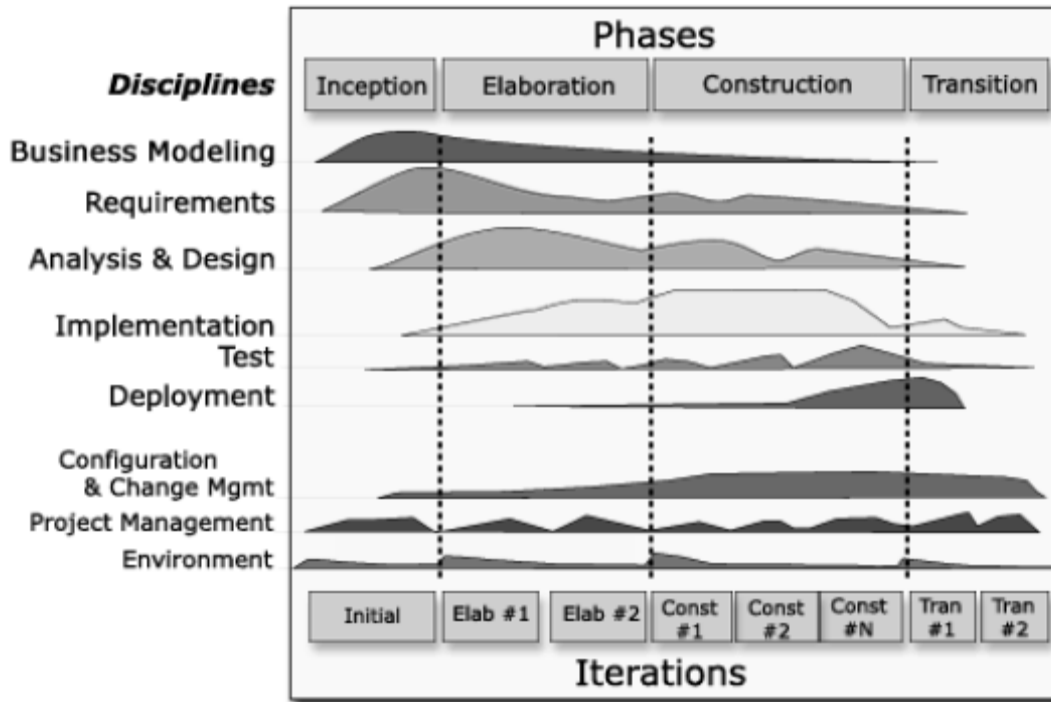


Figure 1: Time and Process Components of RUP

Rational Suite Team Unifying Platform provides integrated tools for managing change, building quality, and communicating results from requirements to release.

3. RATIONAL ROSE Introduction

Rational Rose is a CASE Tool. CASE Stands for *Computer Aided Software Engineering*. They form a category of software that provides a development environment for programming teams. CASE systems offer tools to automate, manage and simplify the development process.

These can include tools for:

- Summarizing initial requirements
- Developing flow diagrams
- Scheduling development tasks
- Preparing documentation
- Controlling software versions
- Developing program code

3.1 Few Other CASE TOOLS

- Berard Object & Class Specifier (BOCS)
- Bridgepoint by Project Technology, Inc.
- Cadre Teamwork by Cadre Technologies, Inc.
- Clyder by Sema Group.
- ICONIX PowerTools by ICONIX Software Engg Inc.
- Methods Workbench by ISDE Metasoft Ltd.
- Objectory Support Environment
- OOAtool by Object International, Inc.
- Paradigm Plus by Platinum Technology, Inc.
- System Architect by Popkin Software & Systems.
- With Class by MicroGold Software Inc.
- Visual Thought by Confluent, Inc.
- Teamwork by Cadre Technologies, Inc.

Rational Rose is the visual modeling software solution that lets you create, analyze, design, view, modify, and manipulate components. You can graphically depict an overview of the behavior of your system with a use-case diagram. Rational Rose provides the collaboration diagram as an alternative to a use-case diagram. It shows object interactions organized around objects and their links to

one another. The statechart diagram provides additional analysis techniques for classes with significant dynamic behavior. A statechart diagram shows the life history of a given class, the events which cause a transition from one state to another and the actions that result from a state change. Activity diagrams provide a way to model a class operation or the workflow of a business process.

Rational Rose provides the notation needed to specify and document the system architecture. The logical architecture is captured in class diagrams that contain the classes and relationships that represent the key abstractions of the system under development. The component architecture is captured in component diagrams that focus on the actual software module organization within the development environment. The deployment architecture is captured in deployment diagrams that map software to processing nodes—showing the configuration of run-time processing elements and their software processes. Rational Rose provides support for two essential elements of modern software engineering: component-based development and controlled iterative development.

While these concepts are conceptually independent, their usage in combination is both natural and beneficial.

Rational Rose's model-diagram architecture facilitates use of the Unified Modeling Language (UML), Component Object Modeling (COM), Object Modeling Technique (OMT), and Booch '93 method for visual modeling. Using semantic information ensures correctness by construction and maintaining consistency.

Rational Rose is currently available in three editions:

- Rose Modeler – no language support
- Rose Professional – support for 1 language
- Rose Enterprise – supports multiple languages including (VC++, VB, Java, and CORBA)

4. History of Rational Rose

Rational Software started in the early 1980s as a developer of software development tools. The product line included the R1000 computer, an Ada-only development computer with an integrated IDE. The company later ported the R1000 software to UNIX and sold the result as "Rational Apex".

In October 1994 James Rumbaugh joined the company and in the fall of 1995 Rational merged with Ivar Jacobson's firm Objectory AB. This created a company which had the three leading software methodology inventors, the "three amigos" (also called "the gang of three"): Grady Booch, James Rumbaugh and Ivar Jacobson in the same house. They started by creating Unified Modeling Language (UML); after that they merged their software development methodologies into Rational Unified Process (RUP).

Rose, a software modeling program, arose from a few engineers formerly at GE in Waukesha, Wisconsin. After Rational acquired the product, it moved much of the development to California.

Rational developed and maintained Rose, afterwards called Rational Rose, as a flagship product.

Rose originated to support Ada programming. It currently supports C++ and Java. Unlike many programming artifacts, which developers retain and maintain, Rose Models merely form a stage in the development of a program; hence designers and programmers can discard them after a few uses, because they can re-generate them from the developed program, using round-trip engineering.

Rose RealTime originated to support the development of complex reactive systems, typically ones written in C, C++ and Java. It combines the Real-Time Object Oriented Modeling (ROOM) method developed by Bran Selic at ObjecTime Corp, and the UML capabilities from Rational Rose. Rose RealTime

supports a model-driven development approach that uses forward engineering to generate, directly from a UML model, up to 90% of the real-time application code found in telecommunications switches and industrial controllers. Object Time developed the original product in Kanata, Canada prior to its acquisition by Rational Software on December 14, 1999.

IBM acquired Rational in February 2003 and incorporated it into the IBM Software Group Division where it became the fifth brand, alongside Websphere, Tivoli, DB2, and Lotus.

5. Rational Rose Features

Rational Rose provides the following features to facilitate the analysis, design, and iterative construction of your applications:

- Use-Case Analysis
- Object-Oriented Modeling
- User-Configurable Support for UML, COM, OMT, and Booch '93
- Semantic Checking
- Support for Controlled Iterative Development
- Round-Trip Engineering
- Parallel Multiuser Development Through Repository and Private Support
- Integration with Data Modeling Tools
- Documentation Generation
- Rational Rose Scripting for Integration and Extensibility
- OLE Linking
- OLE Automation
- Multiple Platform Availability

6. Visual Modeling

Visual modeling is the mapping of real world processes of a system to a graphical representation. Models are useful for understanding problems, communicating with everyone involved with the project (customers, domain experts, analysts, designers, etc.), modeling complex systems, preparing documentation, and designing programs and databases. Modeling promotes better understanding of requirements, cleaner designs, and more maintainable systems.

A model is an ideal way to portray the abstractions of a complex problem by filtering out nonessential details. The developer must abstract different views or blueprints of the system, build models using precise notations, verify that the models satisfy the requirements of the system, and gradually add detail to transform the models into an implementation. The models are designed to meet the needs of a specific audience or task, thereby making them more understandable and manageable.

Visual modeling has one communication standard: the Unified Modeling Language (UML). The UML provides a smooth transition between the business domain and the computer domain. Using the UML, all members of a design team can work with a common vocabulary, minimizing miscommunication and increasing efficiency.

Visual modeling captures business processes by defining the software system requirements from the user's perspective. This streamlines the design and development process. Visual modeling also defines architecture by providing the capability to capture the logical software architecture independent of the software language. This method provides flexibility to your system design since the logical architecture can always be mapped to a different software language. Finally, with visual modeling, you can reuse parts of a system or an application by creating components of your design. These components can then be shared and reused by different members of a team allowing changes to be easily incorporated into already existing development software.

7. UML (Unified Modeling Language)

7.1 History of UML

During 1990s many different methodologies existed, along with their own set of notations. Three most popular methods were OMT, Booch and Jacobson. Each had its own pros and cons.

OMT: Strong in analysis and weaker in design area

Booch: Strong in design and weaker in analysis

Jacobson: Strong in Behavior analysis and weaker in the other areas.

Due to simultaneous existence of many methodologies a 'Notation War' started, where a same symbol had different meaning in different methodologies. Thus portability was hard to achieve and whole development world was divided into sub areas each operating in different Notations.

Example: A filled circle was multiplicity indicator in OMT and an aggregation symbol in Booch.

Grady Booch and James Rumbaugh at Rational Software Corporation started work on UML in 1994. Their goal was to create a new method "Unified Method" that would unite the Booch method and OMT-2 method. In 1995 Ivar Jacobson, the man behind OOSE and Objectory method joined them in their work. The first public draft (ver 0.8) introduced in Oct 1995. They released the version 1.0 in Jan 1997. Rational Software formed UML partner consortium and were joined by companies like Digital Equipment Corporation, Oracle, Microsoft, Texas Instruments, Hewlet Packard, Unisys, MCI System House, ICON Computing, IBM and Intellicorp. UML represents the unification of the Booch, OMT and Objectory notations. UML provided basis for de-facto standard in the domain of object-oriented analysis and design.

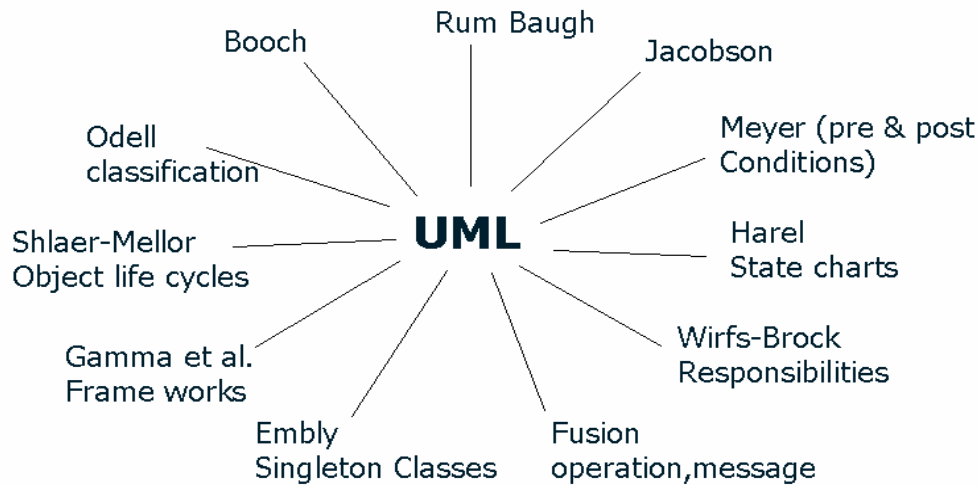


Figure 2: UML Unification

7.2 UML Description

UML stands for Unified Modeling Language. It is a general purpose visual modeling language which is used to specify, visualize, construct and document the artifacts of an object-oriented system under development. It is not just a notation for drawing diagrams, but a complete language for capturing knowledge about a subject and expressing knowledge regarding the subject for the purpose of communication. It has static, dynamic, environment and organizational parts, and is based on object-oriented paradigm. It is intended to be supported by interactive visual modeling tools, which have code generators and report writers.

UML captures information about, both the static structure as well as the behavior of the system. The structure defines the kinds of objects important to a system as well as the relationships among the objects. The dynamic behavior defines the history of objects over time and the communicators among the objects to accomplish the goals.

7.3 Goals of UML

The goals of UML are to

1. Be a ready to use simple and an extensible visual modeling language.
2. Formalize a set of concepts which constitute the object oriented paradigm. This set of concepts can be extended depending on varying interpretations without having to repeatedly redefine the fundamental concepts.
3. Allow adding new concepts and notations beyond the core.
4. Allow variant interpretations of existing concepts when there is no clear consensus.
5. Allow specialization of concepts, notation and constraints for particular domains.
6. Be implementation as well as process independent.

7. Support higher-level concepts like collaborations, frameworks, patterns and components.
8. Be widely applicable and integrate the best processes.
9. Expressive enough to handle all the concepts of that arise in modern system such as concurrency and distribution as well as software engineering mechanisms, such as encapsulation and components.
10. Be as simple as possible while still being capable of modeling the full range of practical systems that need to be built.
11. Address recurring complexity problems using component technology, visual programming, pattern and frameworks.

7.4 UML STRUCTURE

A system is described with a number of different aspects,

- Functional (its static structure and dynamic interaction)
- Non Functional (timing requirements, reliability, deployment etc)
- Organizational aspects (work organizational, mapping to code modules)

Thus a system is described in a number of views, where each view represents a projection of the complete system description, showing a particular aspect of the system. Each view is described in a number of diagrams that contain information emphasizing a particular aspect of the system at a time. A diagram contains graphical symbols that, represents the model elements of system. The various views of UML are shown below:

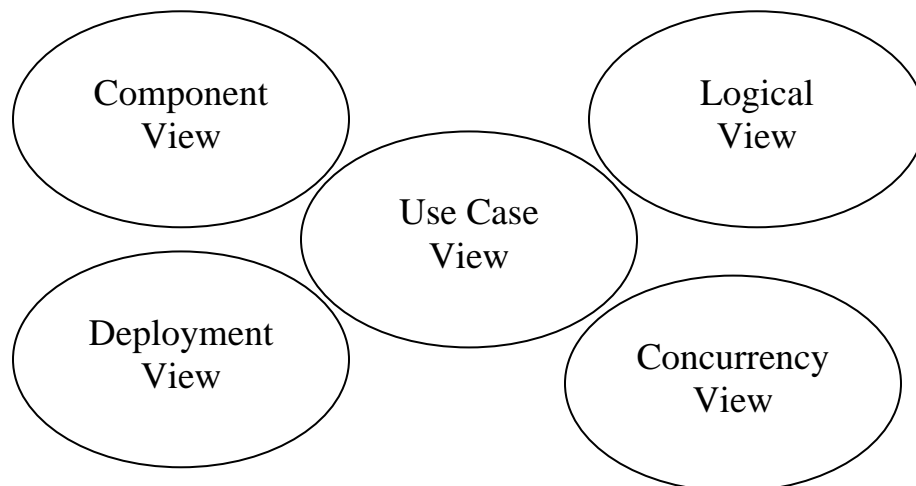


Figure 3: UML Views

- **Use-Case View:** Models functionality of the system, as perceived by external users known as actors. Purpose is to list the actors and use-cases, and show which actor participates in each use case. It is for customers, designers, developers and testers.
- **Logical View:** Describes how the system functionality is provided. Mainly for designers and developers. It looks inside the system and describes both the static structure and the dynamic collaborations that occur when the objects send messages to each other.
- **Component View:** Description of the implementation modules and their dependencies. Mainly for developers. Also known as implementation model view or the development view.
- **Concurrency View:** Deals with division of system into process and processors. Efficient resource usage, parallel execution and the handling of asynchronous events.
- **Deployment View:** Shows Physical Deployment of the system such as the computers, devices and connection b/w them. Mainly for developers, integrators and testers.

8. Modeling with Rational Rose

Independent of Frameworks, you can use Rational Rose's graphical user interface to display, create, modify, manipulate, and document the elements in a model using these windows:

- Application window
- Browser window
- Documentation window
- Diagram window
- Overview window
- Specification window
- Log window

Rational Rose displays the diagram, specification, and documentation windows within the application window. The log window is a dock able window you can move, dock or undock, or close.

8.1 Application Window

An application window contains a title bar, menu bar, toolbar, and a work area where the toolbox, browser, documentation window, diagram window, and specification window appear.

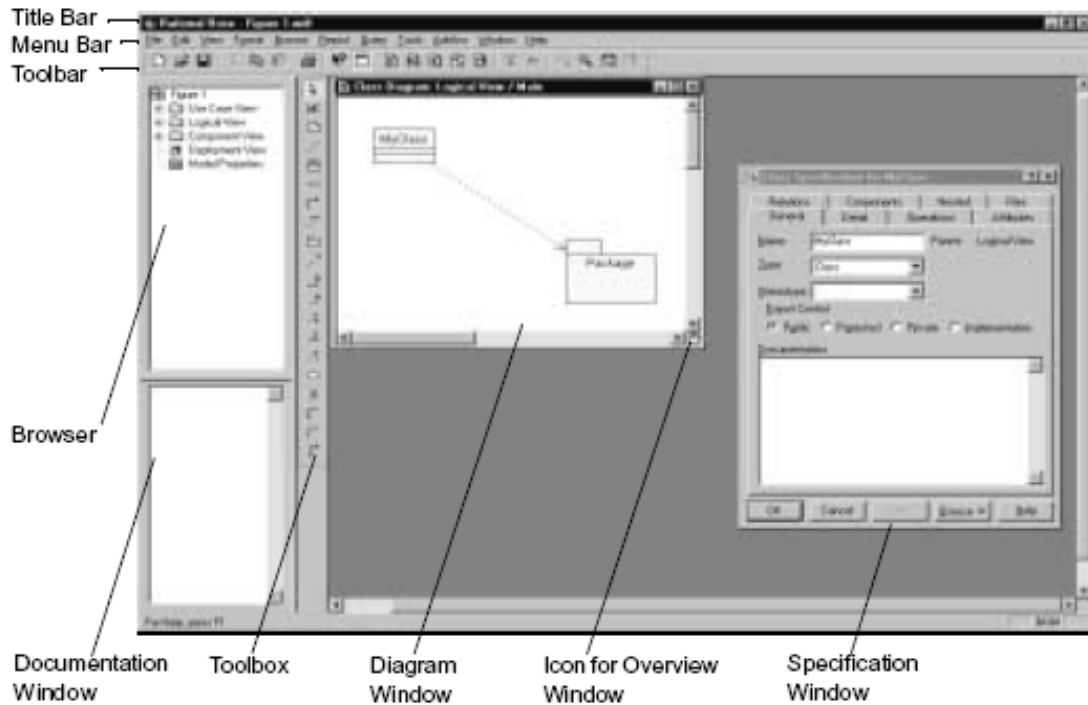


Figure 4: Application Window

8.1.1 Title Bar

The title bar always displays the diagram type. Additional information (like the view or diagram name) is often displayed depending on the diagram/model being viewed. The title bar includes a Control-Menu box, Minimize button, Restore button, and Close button.

Control-Menu Box

Clicking the Control-Menu box (on the application or diagram window) displays a menu with the following commands:

- **Restore** Restores focus to that diagram window.
- **Move** Highlights the border of the window. Move your pointer to the Title Bar, click and drag the window to the desired location.
- **Size** Highlights the border of the window. Move your pointer to the border and resize the window as desired.
- **Minimize** Reduces the window to an icon placing it in the bottom of the application window.
- **Maximize** Enlarges the window to fit the entire screen.
- **Close** Closes the window.

Minimize, Restore, and Close Buttons

These buttons allow you to minimize, restore, or close the diagram or application window.

8.1.2 Menu Bar

The menu bar changes depending on which diagram you are working.

8.1.3 Toolbar

The standard toolbar is displayed directly under the menu bar, along the top of the application window. This toolbar is independent of the open diagram window.

The following icons are available for use on the standard toolbar.



Figure 5: Application window toolbar



New Model

Clicking the **New Model** icon creates a new model.



Open Model

Clicking the **Open Model** icon opens the **Load Model** dialog box.

You can open a model from anywhere within the design.

If you have a model open when you click either the **New** or **Open** icon, you are prompted to save your current model. Clicking **No** discards all changes since your last save. Clicking **Yes** saves your changes and either opens a new model or displays the **Load Model** dialog box.



Save Model or Log

Clicking the **Save Model** icon opens the **Save Model to** dialog box.

Enter a new file name. After the model is named and saved, clicking this icon automatically saves your changes to the current model without displaying the dialog box. This will also save the log if the log window is open.



Cut

Clicking the **Cut** icon removes icons from your model. Element(s) must be selected to activate the icon. Cutting an element will also cut associated relationships. You can cut multiple selected items.



Copy

Clicking the **Copy** icon copies an element to a new location on the same model, or to a new model, without affecting the original model.



Paste

Clicking the **Paste** icon pastes a previously cut or copied element on the Clipboard onto another location.



Print Diagrams

Clicking the **Print** icon prints diagrams to the default printer.



Context Sensitive Help

Clicking the **Context Sensitive Help** icon makes all topics covered in the online Help available. Click this icon and then click the item with which you want help.



View Documentation

Clicking the **View Documentation** icon displays the documentation window on the diagram.



Browse Class Diagram

Clicking the **Browse Class Diagram** icon opens the **Select Class Diagram** dialog box.



Browse Interaction Diagram

Clicking the **Browse Interaction Diagram** icon opens the **Select Interaction Diagram** dialog box.



Browse Component Diagram

Clicking the **Browse Component Diagram** icon opens the **Select Component Diagram** dialog box.



Browse State Machine Diagram

Clicking the **Browse State Machine Diagram** icon opens the **Select State chart Diagram** or **Activity Diagram** dialog box.



Browse Deployment Diagram

Clicking the **Browse Deployment Diagram** icon opens the **Deployment Diagram** dialog box.



Browse Use-Case Diagram

Clicking the **Browse Use-Case Diagram** icon opens the **Selected Use Case Diagram** dialog box.



Browse Parent

Clicking the **Browse Parent** icon displays the “parent” of the selected diagram or specification. If you have a specification selected, the specification for the parent of the “named” item is displayed.



Browse Previous Diagram

Clicking the **Browse Previous Diagram** icon displays the last displayed diagram.



Zoom In

Clicking the **Zoom In** icon magnifies the current diagram to view an area in detail.



Zoom Out

Clicking the **Zoom Out** icon minimizes the current diagram allowing you to view more information.



Fit in Window

Clicking the **Fit in Window** icon centers and displays a diagram within the limits of the window. This command changes the zoom factor so that the entire diagram appears.

8.1.4 Toolbox

The diagram toolbox consists of tools that are appropriate for the current diagram.

Changing diagrams automatically displays the appropriate toolbox. When a modifiable diagram window is active, a toolbox with tools appropriate for the current diagram is displayed. If the current diagram is contained by a controlled unit or the model is write-protected, the toolbox is not displayed. While each diagram has a set of tools applicable for the current diagram, all toolboxes have the **Selector**, **Separator**, and **Lock** icons.

Selector Icon

The selector icon is used to select icons on the diagram. This icon cannot be removed from the toolbox.

Separator Icon

The separator icon is used to put a small space between icons on the toolbox. You can have as many separators as you want, but you must have at least one.

Lock Icon

The lock icon can be set to locked or unlocked. In the locked mode, any tool icon stays in the selected state until the diagram loses focus or another tool button is selected. This option facilitates the rapid placement of several identical icons without repeatedly returning to the diagram toolbox.

You can obtain the lock functionality without the icon through the shortcut menu or by pressing the SHIFT key while placing an element. Releasing the SHIFT deactivates the lock feature.

8.2 Browser Window

The browser is a hierarchical navigational tool that allows you to view the names and icons of interaction, class, use case, statechart, activity, and deployment diagrams as well as many other model elements. When a class or interface is assigned to a component, the browser displays the assigned component name in an extended name. The extended name is a comma-separated list within parenthesis to the right of the class and interface name. The extended list includes all the assigned components.

8.3 Documentation Window

The documentation window is used to describe model elements or relationships. The description can include such information as the roles, keys, constraints, purpose, and essential behavior of the element. You can type information either here or through the documentation field of a specification.

To view the documentation window, click **View > Documentation**. A check mark next to documentation indicates the window is open.

Only one documentation window can be open at a time, but as you select different items, the window will be updated accordingly. When the window is first displayed, it will be docked to the lower left corner of the Rose application window. To move the window, click and drag on its border. The window outline indicates the window state: a thin, crisp line indicates the window will be docked, while a thicker, hash mark-type border indicates it will be floating.

Characteristics of the docked and floating states of the window are as follows:

Docked

The window can be moved within the dockable region of the model, but it remains positioned along the border. The size remains fixed. The title can be displayed through a tool tip (place your pointer anywhere in the window).

The window may be docked at any time.

Floating

The window can be moved to any location and is always displayed on top of the diagram. Size can be changed by clicking and dragging along the border in a vertical or horizontal direction. The window title displays the type (class or object) and name of the class or object.

8.4 Log Window

Rose uses the log window to report progress, results, and errors that occur as a result of a command or action in your model. The messages posted to the log are prefixed with a time stamp, enabling you to track when an event or action occurred. Like the documentation window, the log window can be docked or floating. You can dock or undock the window by right-clicking anywhere in the window and toggling **Allow Docking** option. When docked, the log window is positioned along the border of the application window. If docking is not enabled or if you drag the window outside of the application frame, the window is floating. A floating window is always on top. In addition, you can hide the log window by right-clicking anywhere in the window and clicking **Hide**.

To redisplay the window, click **View > Log**.

You can save the contents of the log window to a file as well as clear the log contents. To save the log, click **File > Save LogAs**.

To clear the log, right-click anywhere in the log window and click **Clear**.

8.5 Diagram Window

Diagram windows allow you to create and modify graphical views of the current model. Each icon in a diagram represents an element in the model. Since diagrams are used to illustrate multiple views of a model, each model element can appear in none, one, or several of a model's diagrams. This means you can control which elements and properties appear on each diagram.

Diagrams are contained by the model elements, they represent:


- A logical package (also User Services, Business Services, and Data Services) contains an automatically created class diagram called "Package Overview," and user created class diagrams, collaboration diagrams, interaction diagrams, and three-tiered diagrams.
- A component package contains component diagrams.
- A class contains its state diagrams.
- A model contains the diagram for its top level components, its three-tiered service model diagram, its deployment diagram, and the diagram contained by its logical package and component packages. These top-level components can be classes, components, devices, connections, and processors.

8.6 Overview Window

The overview window is a navigational tool that helps you move to any location on all Rational Rose diagrams. When a diagram is larger than the viewable area within the diagram window, it is not possible to see the whole diagram without scrolling.

The overview window provides a scaled-down view of the current diagram so you can see the entire diagram.

To move to an exact area of your diagram, use the following steps:

- 1** Move the pointer over the hand  located in the lower, right side of the diagram window. Notice how the pointer appears as a + when the pointer is located over the active hand.
- 2** Click on the hand icon so the overview window appears.
- 3** Hold down the mouse button and move the box inside the overview window to a desired diagram location.

Specification Window

A specification enables you to display and modify the properties and relationships of a model element, such as a class, a relationship, an operation, or an activity. The information in a specification is presented textually; some of this information can also be displayed inside icons representing the model element in diagrams.

You can change properties or relationships by editing the specification or modifying the icon on the diagram. The associated diagram or specification is automatically updated.

To display a specification:

1. Right-click the icon in either the diagram or browser, and then click **Open Specification** from the shortcut menu.
2. Click the icon in either the diagram or browser, and then click **Browse > Specification**.
3. Double-click on the icon in either the diagram or browser. (If you have selected the **Double-Click to Diagram** option in the **Options** dialog box, a diagram may appear instead of a specification.)

Printing Diagrams and Specifications

The **Print** dialog box allows you to print diagrams and specifications. Table below describes the tabs in the **Print** dialog box.

Tab	Description
General	Allows you to specify a printer, a selection of diagrams and specifications, and the number of copies to be printed.
Diagrams	Allows you to select and view a list of diagrams to be printed.
Specifications	Allows you to select and view a list of specifications to be printed.

Tab	Description
Layout	Allows you to select layout settings for printing diagrams and specifications.

Print Preview

The print preview option allows you to see how a diagram will appear when printed. Also, print preview displays the total number of pages the diagram will take to print on the status bar.

Zoom In/ Zoom Out

Click either **Zoom In** or **Zoom Out** to view a diagram at different magnified sizes. Also, you can click on any part of the diagram to get a magnified view.

Print

Click **Print** to display the Print dialog box.

One Page / Two Pages

Click **Two Page** to display the diagram in two pages or click **One Page** to view the diagram in one page. When diagrams are viewed in two pages, the Next Page button becomes active and enables you to view other pages. The Previous Page button becomes active when there is a previous page to view.

Close

Click **Close** to return to an active window.

Apply Filter Dialog Box

The **Apply Filter** dialog box allows you to search for diagrams and specifications within your model. The filter is especially useful when you print diagrams from

large models. To print a specific diagram in a model, type in the name, type, or path of the diagram you are trying to print.

Name

Provides a list of all diagram names depending on search criteria.

Type

Provides a list of all diagram types depending on search criteria.

Path

Provides a list of each path for diagrams displayed. Next, press the OK button to locate the diagram. Then, with the diagram selected, press OK from the Print dialog box to print the diagram.

To search for a diagram or a specification in the **Apply Filter** dialog box, you can use the * (asterisk) wildcard character. For example:

- A* matches any name beginning with the letter A
- *A matches any name ending with the letter A
- *A* matches any name containing the letter A

Saving in Various Formats

If you want to save a Rational Rose model as a different format, you may select any of the following options from the **Save As Type** list in the **Save Model To** dialog box:

- Models *.mdl (the current version of Rose)
- Petal *.ptl
- Rose 6.1/6.5 Model
- Rose 4.5/6.5 Model
- Rose 4.0 Model
- Rose 3.0 Model

9. Introduction to Diagrams

Diagrams are views of the information contained in a model. Rational Rose automatically maintains consistency between the diagram and its specifications. You can change properties or relationships by editing the specification or modifying the icon on the diagram. The associated diagrams or specifications are automatically updated. In a diagram window, you can create and modify graphical views of the model.

Rational Rose supports the following kinds of diagrams:

- Class diagram
- Object diagram
- Use-case diagram
- Collaboration diagram

- Sequence diagram
- Component diagram
- Statechart diagram
- Deployment diagram
- Activity diagram

Each icon on a diagram represents an element in the model. Since diagrams illustrate multiple views of a model, each model element can appear in none, one, or several of a model's diagrams. You can control which elements and properties appear on each diagram.

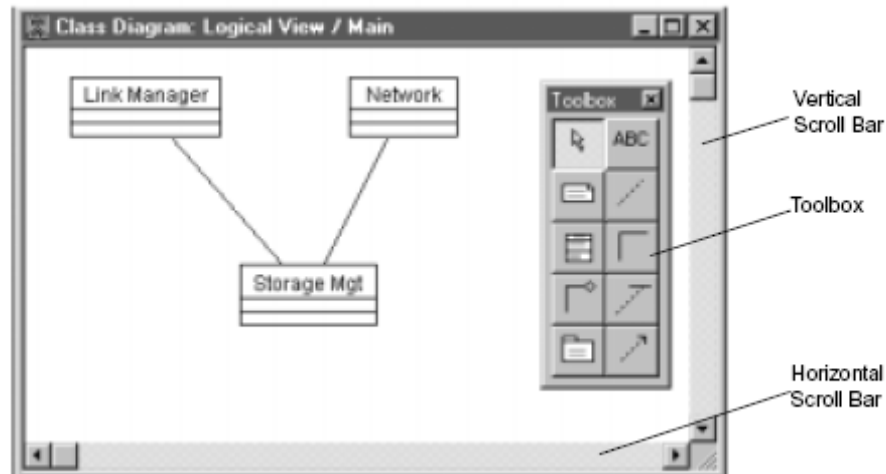


Figure 6: Diagram Window

You can display multiple diagrams simultaneously in the application window. To display diagrams in cascaded windows click **Window > Cascade** or **Tile**.

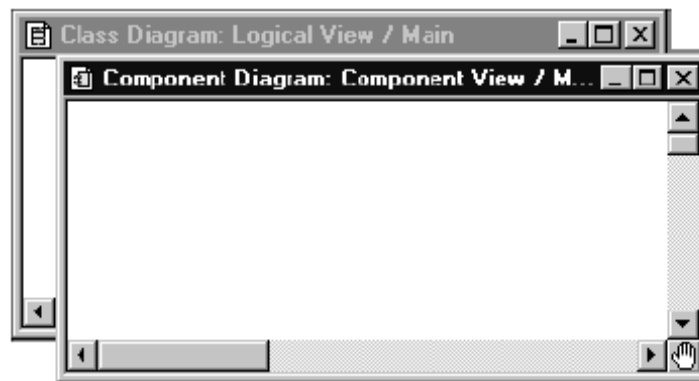


Figure 7: Multiple Diagrams—Cascade Windows

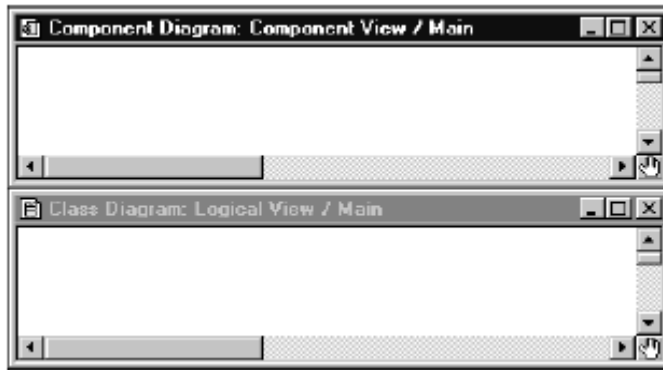


Figure 8: Multiple Diagrams—Tiled Windows

Overloading also allows you to do multi-lingual, component-based development. For example, an application can be modeled even if the GUI for screen input is in VB or Java; the processing is in C++, and the database in Oracle. In this example, each application can have its definition of a class “Customer” do different things.

9.1 Deleting Model Elements

There are two ways to delete model elements in Rational Rose: you can perform a shallow delete or a deep delete. A shallow delete removes the element icon from a diagram. A deep delete removes model elements from a model completely.

Shallow Delete

A shallow delete is useful when you want to remove a model element icon from a diagram but keep the model element in the model. A shallow delete keeps the model element in the browser and removes the icon of the element from the diagram. If you perform a shallow delete on an element without a name, Rational Rose will delete the model element completely from the model.

To perform a shallow delete on a selected model element that appears on a diagram:

- Click **Edit > Delete**
- Press DELETE.

Deep Delete

A deep delete is useful when you want to remove a model element completely from a model.

To perform a deep delete on a selected diagram model element(s):

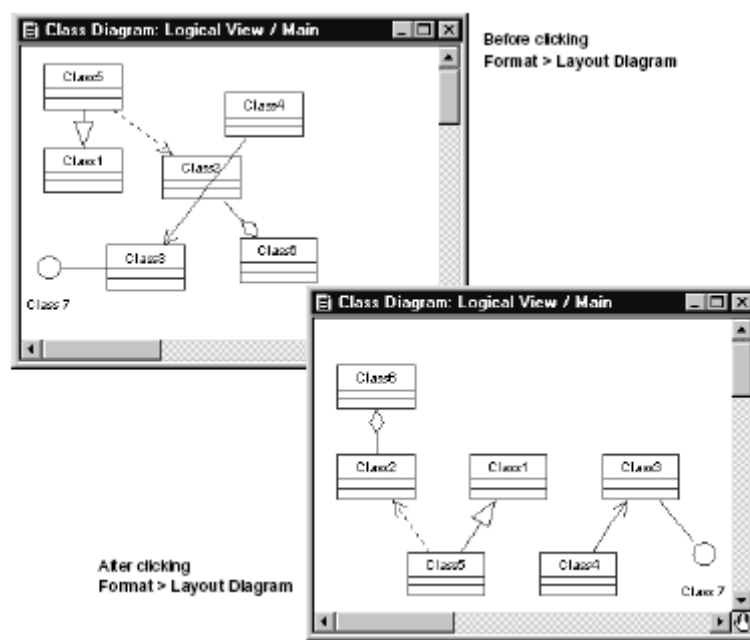
- Click **Edit > Delete from Model**.
- Press CTRL + D.
- Right-click an element in the browser and then click **Delete** from the shortcut menu.

9.2 Correlations

Depending on the diagram selected, a correlation can be a relationship, a link, a dependency, a transition, or a connection. The word correlation can stand for any of the items previously listed.

9.3 Laying Out a Diagram

When a diagram contains many elements (also called shapes) and many relationships (also called correlations), it can become difficult to read. The layout diagram feature is designed to make a diagram easier to read by rearranging elements on a diagram to clarify their relationships. This is done by minimizing the number of crossed links and positioning shapes in an order that reflects their relationships.



10. Class/Object Diagrams

10.1 Overview

A class diagram is a picture for describing generic descriptions of possible systems.

Class diagram models static view of the system. It is the main diagram which identifies the main classes (defines rules about objects) and the relationship among these classes. This diagram is very important because as it is used as a source for code generation or design stage. A class integrates the static (data) and dynamics (behavior) into one cohesive unit. Class diagrams contain classes and

object diagrams contain objects, but it is possible to mix classes and objects when dealing with various kinds of metadata, so the separation is not rigid.

Object diagram is a special type of class diagram showing instance of classes i.e. objects and can be used to test or understand a class diagram.

UML notation to represent a class is a rectangle consisting of three parts:

- Class Name
- Attributes
- Operations

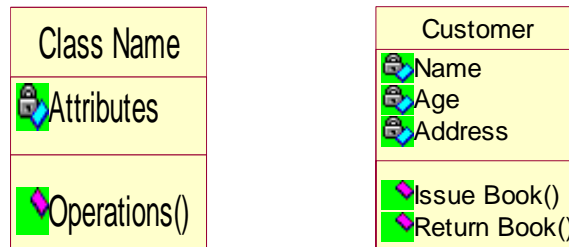


Fig: Class Notation

Additionally you can also specify the visibility of the attributes and operations i.e. which other objects can see the attributes and operations.

This typically includes public (+), private (-), protected (#) and package (~) as visibility values. While writing operation specification, it must specify all arguments and their data types (separated by colon), return data type, its visibility and the constraints within { }.

Three types of relations via- association, aggregation and generalization can exist between classes in the class diagram.

10.2 Association

It represents a semantics relationship between two classes. It relates a class A to class B and is shown as a link between classes. While reading textual description of requirements they correspond to verbs and classes correspond to nouns.

Multiplicity (number of participating objects), role and constraints are also shown in the association. Constraints are shown between { } below the class icons.

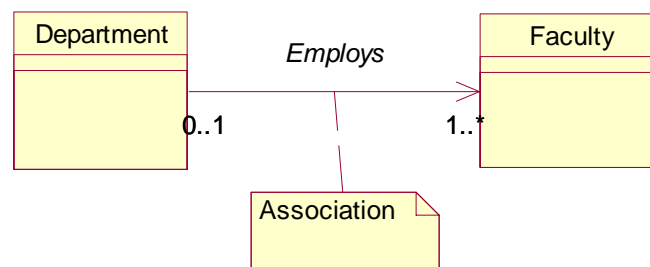


Figure 10: Association example

10.3 Aggregation

It is a special type of association which supports building complex objects. For example different components can be assembled to make a car as shown

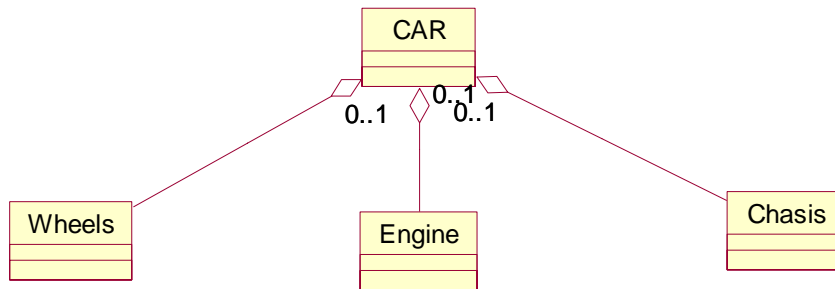


Figure 11: Aggregation relationship

10.4 Generalization

Is-a relationship is shown by generalization. The classes between which is-a relationship is identified are called superclass and subclass. Subclass inherits the features of superclass in addition to which are specific to subclass.

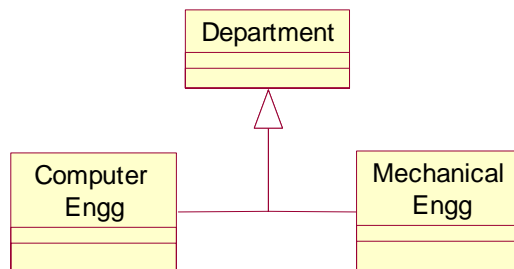


Figure 11: Generalization in classes

Class diagrams contain icons representing classes, interfaces, and their relationships. You can create one or more class diagrams to depict the classes at the top level of the current model. Such class diagrams are themselves contained by the top level of the current model. You can also create one or more class diagrams to depict classes contained by each package in your model. Such class diagrams are themselves contained by the package enclosing the classes they depict. The icons represent logical packages and classes in class diagrams. You can change properties or relationships by editing the specification or modifying the icon in the diagram. The associated diagrams or specifications are automatically updated.

10.5 Types of Classes

10.5.1 Entity Class

An entity class models information and associated behavior that in generally long lived. This type of class may reflect a real-world entity or it may be needed to

perform tasks internal to system. Many times they are application independent, meaning that they may be used in more than one application. Entity classes are typically found early in the elaboration phase. They are often called “Domain” classes since they usually deal with abstractions of real world.

10.5.2 Boundary Class

Boundary classes handle the communication between the system surroundings and the inside of the system. They can provide the interface to the user or another system (i.e. the interface to an actor). They constitute the surrounding dependent part of the system. Boundary classes are used to model the system interfaces. They are also added to facilitate communication with the other system.

10.5.3 Control Class

Control class model sequencing behavior specific to one or more use cases. Control classes coordinate the events needed to realize the behavior specified in the use case. They are thought to be the running or executing the use case- they represent the dynamics of the use case. Control classes are typically application dependent classes. In early phases of development a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use case.

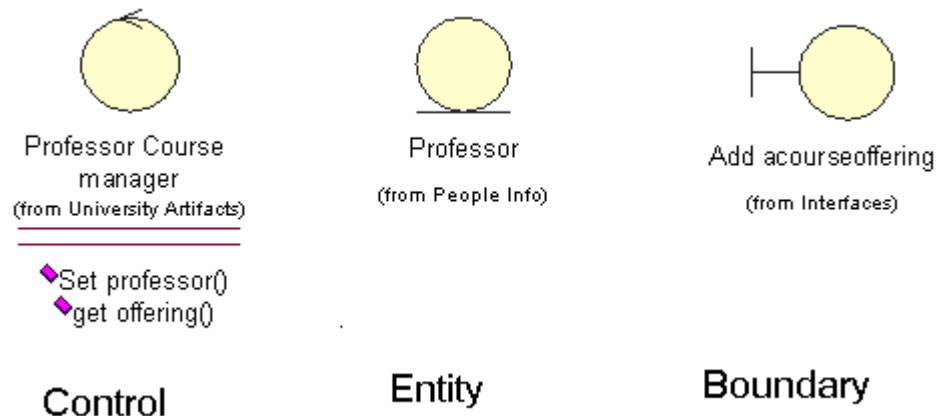


Figure 12: Control, Entity and Boundary Class

10.6 Package

A package in the logical view of the model is a collection of related packages and/or classes. By grouping classes into packages, we can look at the higher level view of the model (i.e. the package) or you can dig deeper into the model by looking at what is contained by the package. Each package contains an interface that is realized by its set of public classes – those classes to which classes in the other packages talk. If the system is complex, package may be created early in the Elaboration Phase to facilitate communication.



Figure 13: Packages

10.7 Class Diagram Toolbox

The application window displays the following toolbox when the current window contains a class diagram, you have selected **View > As Unified**, and you have customized the toolbox to display all the tool options.

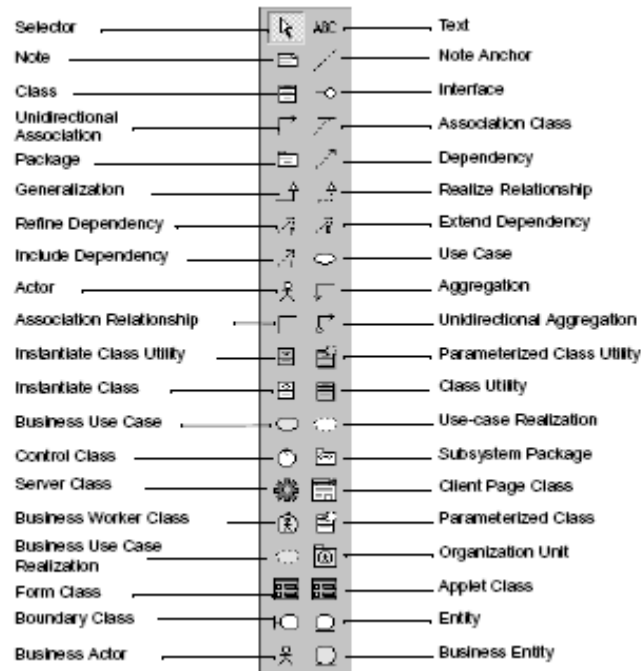


Figure 14: Class Toolbox

Creating and Displaying a Class Diagram

You can create or display a class diagram in one of three ways:

- Click **Browse > Class Diagram**.
- On the toolbar, click the class diagram icon.
- On the browser, double-click the class diagram icon.

Adding and Hiding Classes and Filtering Class Relationships

The commands on the **Query** menu allow you to control which model elements are represented by icons in the current diagram.

On the **Query** menu, clicking:

- **Add Classes** adds classes to the diagram by name.
- **Add Use Cases** adds use cases to the diagram by name.

- **Expand Selected Elements** adds classes to the diagram based on their relationships to selected classes.
- **Hide Selected Elements** removes selected classes from the diagram and optionally removes their clients or suppliers from the diagram.
- **Filter Relationships** controls which kinds of relationships appear in the diagram.

10.8 Class Specification

A **Class Specification** displays and modifies class properties and relationships. Some of the information in the specification can also be displayed inside class icons. If a field does not apply to a particular class type, the field is unavailable and you cannot add or change information in the field.

To display a **Class Specification**, click an icon representing the class in a class diagram and click **Browse > Specification**.

The **Class Specification** consists of the following tabs:

General, Detail, Operations, Attributes, Relations, Component, Nested, and Files.

Type

Your **Type** choices include: Class, Parameterized Class, Instantiated Class, Class Utility, Parameterized Class Utility, Instantiated Class Utility, and Metaclass.

Parent

The parent to which the class belongs (its package) is displayed in this static field.

Stereotype

A stereotype represents the subclassification of an element. It represents a class within the UML metamodel itself; that is, a type of modeling element. Some stereotypes are already predefined. You can also define your own stereotypes. Stereotypes can be shown in the browser and on diagrams. The name of the stereotype may appear in angle brackets <<>>, depending on the settings found in either the **Diagram** or **Browser** tabs of the **Options** dialog box. Click **Tools > Options** to display the **Options** dialog box.

To show stereotypes on the diagrams, right-click a class, and then click **Options > Stereotype Display > None, Label, Decoration, or Icon** from the shortcut menu.

Export Control

The **Export Control** field specifies how a class and its elements are viewed outside of the defined package.

Option	Description
Public	The element is visible outside of the enclosing package and you can import it to other portions of your model. Operations are accessible to all clients.
Protected	The element is accessible only to subclasses, friends, or the class itself.
Private	The element is accessible only to its friends or to the class itself.

Table 3: Export Options

The **Export Control** field can be set only in the specification. No special annotation is related to access control properties.

To change the export control type for the class, click the appropriate option in the **Export Control** field. You can display the implementation export control in the component compartment

10.9 Cardinality

The **Cardinality** field specifies the number of expected instances of the class. In the case of relationships, this field indicates the number of links between each instance of the client class and the instance of the supplier. You can set a specific cardinality value for the client class, supplier class, or both.

Use the following syntax to express cardinality.

Type	Description
n (default)	Unlimited number of instances
1	One instance only
0..n	Zero or more instances
1..n	One or more instances
0..1	Zero or one instance
<literal> ^a	Exact number of instances
<literal>..n	Exact number or more instances
<literal>..<literal>	Specified range of instances
<literal>..<literal>,<literal>	The number of instances will be in the specified range or an exact number of instances
<literal>..<literal>, <literal>..<literal>	The number of instances will be in one of the specified ranges

a. Where <literal> is any integer greater than or equal to one.

Table 4: Cardinality Options

To display class cardinality on an icon, right-click the icon and select a cardinality through the shortcut menu. A literal value can only be specified on the specification.

10.10 Persistence

Persistence defines the lifetime of the instances of a class. A persistent element is expected to have a life span beyond that of the program or one that is shared with other threads of control or other processes. Use this field to identify the persistence for elements of this class.

The persistence of an element must be compatible with the persistence that you specified for its class. If class persistence is set to **Persistent**, then the object is

persistent, static, or transient. If class persistence is set to **Transient**, then the object persistence is either static or transient.

To set the persistence, click the applicable option in the **Persistence** field. You can display the persistence in the diagram by clicking **Show Persistence** from the shortcut menu.

10.11 Abstract

The **Abstract** check box identifies a class that serves as a base class. An abstract class defines operations and states that will be inherited by subclasses. This field corresponds to the abstract class adornment displayed inside the class icon. The **Abstract** field is inactive for metaclasses, class utilities, parameterized class utilities, and instantiated class utilities.

10.12 Operations Tab

Operations denote services provided by the class. Operations are methods for accessing and modifying **Class** fields or methods that implement characteristic behaviors of a class. The **Operations** tab lists the operations that are members of this class. Rational Rose stores operation information in an **Operation Specification**. You can access **Operation Specifications** from the **Class Specification** or from the Browser.

The descriptions for each field on the **Operations** tab are discussed below:

- ◆ Public—members of a class are accessible to all clients.
- 🔒 Protected—members of a class are accessible only to subclasses, friends, or to the class itself.
- 🔒 Private—members of a class are accessible only to the class itself or to its friends.
- 🔒 Implemented—the class is accessible only by the implementation of the package containing the class.

10.13 Attributes Tab

The Rational Unified Process asserts that attributes are data values (string or integer) held by objects in a class. Thus, the **Attributes** tab lists attributes defined for the class through the **Class Attribute Specification**. You can add an attribute relationship through **Insert** on the shortcut menu or by pressing the INSERT key. An untitled entry is added. Attributes and relationships created using this technique are added to the model, but do not automatically appear in any diagrams

The descriptions for each field are discussed below:

- ◆ Public—members of a class are accessible to all clients.
- 🔒 Protected—members of a class are accessible only to subclasses, friends, or to the class itself.
- 🔒 Private—members of a class are accessible only to the class itself or to its friends.

 **Implemented**—the class is accessible only by the implementation of the package containing the class.

This **Attribute** tab is active for all class types.

10.14 Nested Tab

A nested class is a class that is enclosed within another class. Classes may contain instances of, inherit from, or use a nested class. Enclosing classes are referred to as parent classes, and a class that lies underneath the parent class is called a nested class.

A nested class is typically used to implement functionality for the parent class. In many designs, a nested class is closely coupled to the parent class and is often not visible outside of the parent class. For example, think of your computer as a parent class and its power supply as a nested class. While the power supply is not visible outside the computer, the task it completes is crucial to the overall functionality of the computer.

10.15 Containment

Physical containment plays a role in the construction and destruction of an aggregate's parts through semantics. The specification of physical containment is necessary for meaningful code generation from the model.

You can set one of the following types of physical containment.

Type	Description
By Value	Physical containment of a value of the part.
By Reference	Physical containment of a pointer or reference to the part.
Unspecified (default)	The type of physical containment has not been specified.

Table 5: Physical Containment Options

To set or change the containment type in the **Relationship Specification**, click the applicable option in the **Containment** field. The application places an adornment at the supplier end of the relationship. You can also select a value from the shortcut menu.

11. Use Case Diagrams

Use-case diagrams present a high-level view of how a system is used as seen from an outsider's (or actor's) perspective. These diagrams graphically depict system behavior (also known as use cases). A use-case diagram may depict all or some of the use cases of a system.

A use-case diagram can contain:

- Actors (“things” outside the system).
- Use cases (system boundaries identifying what the system should do).
- Interactions or relationships between actors and use cases in the system including the associations, dependencies, and generalizations.

Use-case diagrams can be used during analysis to capture the system requirements and understand how the system should work. During the design phase, use-case diagrams can be used to specify the behavior of the system as implemented.

You can create or display a use-case diagram in one of three ways:

- Click **Browse > Use Case Diagram**.
- On the toolbar, double-click the use-case diagram icon.
- In the browser, double-click the use-case diagram icon.

11.1 Actors

Actors represent system users. They help define the system and give a clear picture of what the system should do. It is important to note that an actor interacts with, but has no control over, the use cases.

An actor is someone or something that:

- Interacts with or uses (but is not part of) the system.
- Provides input to and receives information from the system.
- Is external to the system and has no control over the use cases.

Actors are discovered by examining:

- Who directly uses the system?
- Who is responsible for maintaining the system?
- External hardware used by the system.
- Other systems that need to interact with the system.

An actor is a stereotype of a class and is depicted as a “stickman” on a use-case diagram. The name of the actor is displayed below the icon.

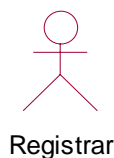


Figure 15: Notation for an Actor

11.2 Use Case

A use case is a sequence of events (transactions) performed by a system in response to a trigger initiated by an actor. A use case contains all the events that can occur between an actor-use case pair, not necessarily the ones that will occur in any particular scenario.

In its simplest form, a use case can be described as a specific way of using the system from a user's (actor's) perspective. A use case also illustrates:

- A pattern of behavior the system exhibits.
- A sequence of related transactions performed by an actor and the system.

Use cases provide a means to:

- Capture system requirements.
- Communicate with the end users and domain experts.
- Test the system.

Use cases are best discovered by examining what the actor needs and defining what the actor will be able to do with the system; this helps ensure that the system will be what the user expects.

The following questions may be used to help identify the use cases for a system:

- What are the tasks of each actor?
- Will any actor create, store, change, remove, or read this information in the system?
- What use case will create, store, change, remove, or read this information?
- Will any actor need to inform the system about sudden, external changes?
- Does any actor need to be informed about certain occurrences in the system?
- What use cases will support and maintain the system?
- Can all functional requirements be performed by the use cases?

Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this use case collection specifies all the ways of using the system.

A use case may have a name, although it is typically not a simple name. It is often written as an informal text description of the actors and the sequences of events between objects. Use case names often start with a verb.

The name of the use case is displayed below the icon.



Select courses to teach

Figure 16: Notation for Use Case

11.3 Flow of Events

A flow of events is a sequence of transactions (or events) performed by the system. They typically contain very detailed information, written in terms of what the system should do, not how the system accomplishes the task. Flows of events are created as separate files or documents in your favorite text editor and then attached or linked to a use case using the **Files** tab of a model element.

A flow of events should include:

- When and how the use case starts and ends
- Use case/actor interactions
- Data needed by the use case
- Normal sequence of events for the use case
- Alternate or exceptional flows

11.4 Relationships

Relationships show interactions between actors and use cases. Association, dependency, and generalization relationships can be drawn from an actor to a use case. The generalize relationship can be drawn between actors. Any association relationships are also presented in a text format on the **Relations** tab for a selected use case or actor.

11.5 Association

An association provides a pathway for communication between use cases and actors. Associations are the most general of all relationships and consequentially, the most semantically weak. If two objects are usually considered independently, the relationship is an association. The association name and its stereotype are typically verbs or verb phrases and are used to identify the type or purpose of the relationship.

There are two different types of associations connected with use-case diagrams:

uni-directional and bi-directional.

- Uni-directional association: By default, associations in use cases are uni-directional and drawn with a single arrow at one end of the association. The end with the arrow indicates who or what is receiving the communication.
- Bi-directional association: To change the communication to be bi-directional, double-click the association to view the **Association Specification**. Click the appropriate **Role A** (or **B**) **Detail** tab, select the

Navigable check box, and click **Apply**. The graphic changes from a line with an arrow at one end to a line with no arrow.

11.6 Dependency

A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning. Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier.

You can connect model elements with dependencies on any diagram except state machine diagrams and object diagrams. For example, you can connect a use case to another use case, a package to another package, and a class to a package.

Dependencies are also used on component diagrams to connect model elements.

11.6.1 Extend Stereotype

An extend relationship is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case. You can place extend stereotypes on all relationships. However, most extend stereotypes are placed on dependencies or associations. Extend relationships are important because they show optional functionality or system behavior.

11.6.2 Include Stereotype

An include relationship is a stereotyped relationship that connects a base use case to an inclusion use case. An include relationship specifies how behavior in the inclusion use case is used by the base use case. Include relationships are important because they represent that the inclusion use case functionality is used by the base use case.

11.6.3 Refine Stereotype

A refine relationship is a stereotyped relationship that connects two or more model elements at different semantic levels or development stages. It represents a fuller specification of something that has already been specified at a certain level of detail. For example, a design class is a refinement of an analysis class. In a refine relationship, the source model element is general and more broadly defined whereas the target model element is more specific and refined.

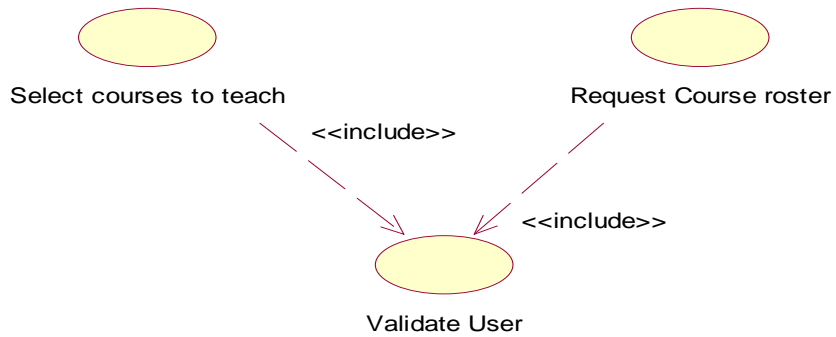


Figure 17: Dependency Display

11.7 Generalization

A generalization relationship is a relationship between a more general class or use case and a more specific class or use case. A generalization is shown as a solid-line path from the more specific element to a more general element. The tip of a generalization is a large hollow triangle pointing to the more general element.

You can place a stereotype on any generalization through the **Generalization Specification**.

11.8 Use-Case Diagram Toolbox

The graphic below shows all the tools that can be placed on the use-case diagram toolbox. Refer to “Customizing the Toolbox” on page 14 for information on adding or deleting diagram toolbox tools. The application window displays the following toolbox when the current window contains a use-case diagram and **As Unified** is selected from the **View** menu.

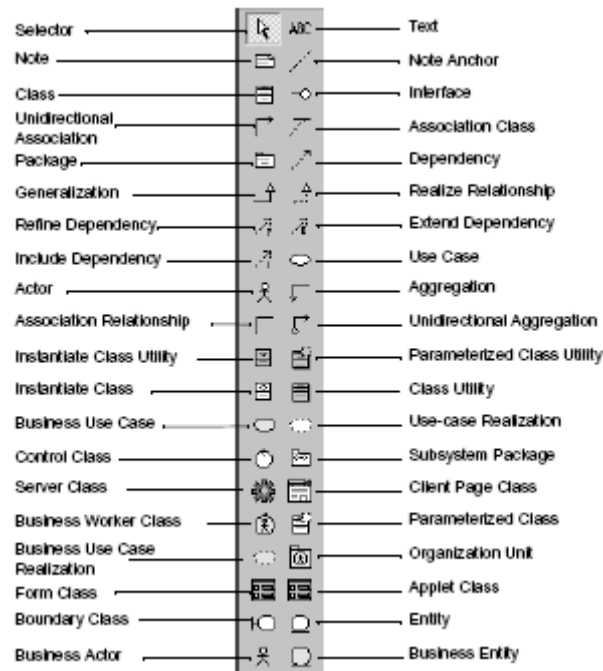


Figure 18: Use Case Toolbox

11.9 Use Case Specification

A **Use-Case Specification** allows you to display and modify the properties and relationships of a use case in the current model.

11.10 Specification Content

The **Use-Case Specification** contains the following tabs: **General**, **Diagram**, **Relations**, and **Files**.

11.11 Name

A use case name is often written as an informal text description of the external actors and the sequences of events between elements that make up the transaction. Use-case/ names often start with a verb. The name can be entered or changed on the specification or directly on the diagram.

11.12 Rank

The **Rank** field prioritizes use cases. For example, you can use the rank field to plan the iteration in the development cycle at which a use case should be implemented.

11.13 Abstract

An abstract notation indicates a use case that exists to capture common functionality between use cases (uses) and to describe extensions to a use case (extends).

12. State Machine Diagrams

A state machine can be defined as a behavior that specifies the valid sequences of activities that an object or interaction goes through during its life in response to the events, together with its responses and actions.

There are two types of state machine diagrams-

- State chart diagram
- Activity diagram

Creating and Displaying a State Machine Diagram

To create a state/activity model:

- 1 Click **Browse > State Machine Diagram**.
- 2 Double-click **New**.
- 3 Name the diagram.
- 4 Specify the type of diagram you want to create: **Activity** or **Statechart**.
- 5 Click **OK**.

12.1 State Machine Specification

A **State Machine Specification** allows you to display and modify the properties and relationships of a state/activity model. A state/activity model contains statechart and activity diagrams. To view the **State Machine Specification**, double-click the state/activity model in the browser. Changes made either through the specification or directly on the icon are automatically updated throughout the model.

The state/activity model icon that appears in the browser can be thought of as a “container” for statechart and activity diagrams and all of their model elements. A state/activity model owns statechart and activity diagrams and is represented semantically with a state machine. A state machine can be defined as a behavior that specifies the valid sequences of activities that an object or interaction goes through during its life in response to events, together with its responses and actions.

Rational Rose automatically creates one state/activity model when you create a statechart or activity diagram. A state/activity model can be relocated to a new owner, such as a class operation or a use case, by dragging it to a new location in the browser. Rational Rose limits you to only one state/activity model per owner.

12.2 State chart diagram

Statechart diagrams model the dynamic behavior of individual classes or any other kind of object. They show the sequences of states that an object goes through, the events that cause a transition from one state or activity to another and the actions that result from a state or activity change.

Statechart diagrams are closely related to activity diagrams. The main difference between the two diagrams is statechart diagrams are state centric, while activity diagrams are activity centric. A statechart diagram is typically

used to model the discrete stages of an object's lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process. Each state represents a named condition during the life of an object during which it satisfies some condition or waits for some event. A statechart diagram typically contains one start state and multiple end states. Transitions connect the various states on the diagram. As with activity diagrams, decisions and synchronizations may also appear on statechart diagrams.

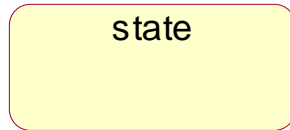


Figure 19: UML Notation of a state

12.2.1 Creating a Statechart Diagram

To create a statechart diagram:

- 1 Click the **Browse State Machine Diagram** button from the toolbar.
- 2 Click **New**.
- 3 Select the **Statechart Diagram** check box in the **New State Machine** dialog box.
- 4 Enter the statechart diagram title.
- 5 Click **OK**.

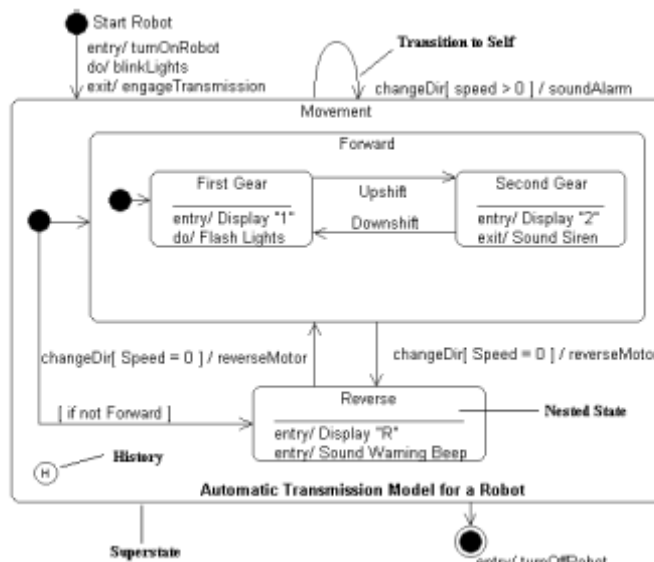


Figure 20: Automatic Transmission Example

12.3 Activity diagram

Activity diagrams provide a way to model the workflow of a business process. You can also use activity diagrams to model code-specific information, such

as a class operation. Activity diagrams are very similar to a flowchart because you can model a workflow from activity to activity. An activity diagram is basically a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities. The main difference between activity diagrams and statecharts is activity diagrams are activity centric, while statecharts are state centric. An activity diagram is typically used for modeling the sequence of activities in a process; whereas, a statechart is better suited to model the discrete stages of an object's lifetime.

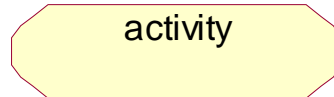


Figure 21: UML Notation of activity

12.3.1 Using Activity Diagrams

Activity diagrams can model many different types of workflows. For example, a company could use activity diagrams to model the flow of approvals for orders or to model the paper trail of invoices. An accounting firm could use activity diagrams to model any number of financial transactions. A software company could use activity diagrams to model a software development process.

12.3.2 Understanding Workflows

Each activity represents the performance of a group of actions in a workflow. Once the activity is complete, the flow of control moves to the next activity or state through a transition. If an outgoing transition is not clearly triggered by an event, then it is triggered by the completion of the contained actions inside the activity. A unique activity diagram feature is a swim lane that defines who or what is responsible for carrying out the activity or state. It is also possible to place objects on activity diagrams. The workflow stops when a transition reaches an end state.

You can attach activity diagrams to most model elements in the use case or logical views. Activity diagrams cannot reside within the component view.

12.3.3 Creating an Activity Diagram

You can create activity diagrams on most model elements except for attributes, associations, or model elements that appear in the component view.

To create an activity diagram:

- 1 In the browser, right-click any model element except for attributes, associations, or model elements that appear in the component view.

2 Click New > Activity Diagram.

3 Rename or double-click to display the **NewDiagram** icon in the browser.

Another way to create an activity diagram:

1 Click the **Browse State Machine Diagram** button from the toolbar.

2 Click **New**.

3 Select the **Activity Diagram** check box in the **New State Machine** dialog box.

4 Enter the activity diagram title.

5 Click **OK**.

12.3.4 Activities

An activity represents the performance of “task” or “duty” in a workflow. It may also represent the execution of a statement in a procedure. An activity is similar to a state, but expresses the intent that there is no significant waiting (for events) in an activity.

12.3.5 Swim lanes

Swim lanes are helpful when modeling a business workflow because they can represent organizational units or roles within a business model. Swim lanes are very similar to objects because they provide a way to tell who is performing a certain role. Swim lanes only appear on activity diagrams. You should place activities within Swim lanes to determine which unit is responsible for carrying out the specific activity. When a swim lane is dragged onto an activity diagram, it becomes a swim lane view. Swim lanes appear as small icons in the browser while swim lane views appear between thin, vertical lines with a header that can be renamed and relocated.

12.3.6 Objects

Rational Rose allows objects on activity, collaboration, and sequence diagrams.

Specific to activity diagrams, objects are model elements that represent something you can feel and touch. It might be helpful to think of objects as the nouns of the activity diagram and activities as the verbs of the activity diagram. Further, objects on activity diagrams allow you to diagram the input and output relationships between activities.

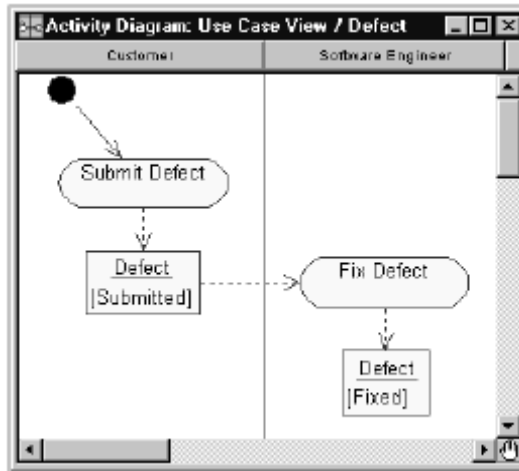


Figure 22: Activity diagram

12.3.7 Object Flow

An object flow on an activity diagram represents the relationship between an activity and the object that creates it (as an output) or uses it (as an input). Rational Rose draws object flows as dashed arrows rather than solid arrows to distinguish them from ordinary transitions. Object flows look identical to dependencies that appear on other diagram types. You do not need a transition if your diagram has two activities connected through an object and two corresponding object flows.

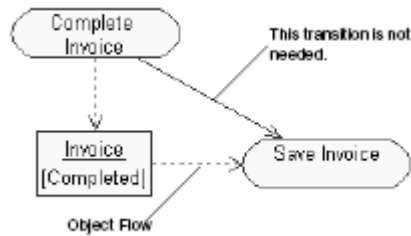


Figure 23: Object Flow

Changing the State of an Object

To change the state of an object on an activity diagram:

- 1 Double-click the object to display the **Object Specification**.
- 2 Select **New** from the **State** list. A new **State Specification** appears.
- 3 Enter descriptive information about the object state in the **State Specification**.
- 4 Click **OK** to close the **State Specification**.
- 5 Click **OK** to close the **Object Specification**.

12.3.8 States

A state represents a condition or situation in the life of an object during which it satisfies some condition or waits for some event. Each state represents the cumulative history of its behavior.

Start and End States

A start state explicitly shows the beginning of a workflow on an activity diagram or the beginning of the events that cause a transition on a statechart. You can have only one start state on a statechart or activity diagram. An end state represents a final or terminal state on an activity diagram or statechart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a statechart diagram.

12.3.9 Transitions

A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states or two activities, or between an activity and a state. Transitions originating from a state cannot have the same event, unless there are conditions on the event. Transitions appear on statechart and activity diagrams.

Transitions are labeled with the following syntax:

event (arguments) [condition] / action ^ target.sendEvent (arguments)

12.3.10 Decisions

A decision represents a specific location on an activity diagram or statechart diagram where the workflow may branch based upon guard conditions. There may be more than two outgoing transitions with different guard conditions but, for the most part, a decision will have only two outgoing transitions determined by a Boolean expression.

12.3.11 Synchronizations

Synchronizations allow you to see a simultaneous workflow in an activity diagram or statechart diagram. They also visually define forks and joins representing parallel workflow.

12.3.12 State and Activity Actions

Each state and activity on a statechart or activity diagram may contain any number of internal actions. An action is best described as a “task” that takes place while inside a state or activity. There are four possible actions within a state or activity:

- On Entry
- On Exit
- Do
- On Event

12.3.13 Guard Condition

Conditional state transitions are triggered only when the conditional expression evaluates to true. Conditions are denoted by surrounding brackets:

Event (args) [condition] / Action ^target.someEvent (args)

To add a condition, click **Guard Condition** on the **State Transition**

Specification and type the conditional expression. You may also include a condition by selecting the event label and changing the text.

13. Interaction Diagrams

An interaction is an important sequence of interactions between objects. Rational Rose provides two alternate views or representations of each interaction—a collaboration and sequence diagram. These are collectively referred to as interaction diagrams. The main difference between sequence and collaboration diagrams is that sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other.

13.1 Creating and Displaying an Interaction Diagram

To create or display a collaboration or sequence diagram:

1 Click **Browse > Interaction Diagram**. The **Select Interaction Diagram** dialog box is displayed.

2 Select a package to “own” the diagram.

3 On the right side of the dialog box, click the diagram name, and then click **OK**.

4 From the **New Interaction Diagram** dialog box, enter the diagram title and click the diagram type. Your choices are **Sequence** or **Collaboration**. Each diagram type is described in detail later in this chapter.

13.2 Collaboration Diagrams

A collaboration diagram is an interaction diagram which shows the sequence of messages that implement an operation or a transaction. These diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object-like entities in the current model.

You can create one or more collaboration diagrams to depict interactions for each logical package in your model. Such collaboration diagrams are themselves contained by the logical package enclosing the objects they depict. During analysis, collaboration diagrams can indicate the semantics of the primary and secondary interactions.

During design, collaboration diagrams can show the semantics of mechanisms in the logical design of the system.

Use collaboration diagrams as the primary vehicle to describe interactions that express your decisions about the behavior of the system. They can also be used to trace the execution of a scenario by capturing the sequential and parallel interaction of a cooperating set of objects. Collaboration diagrams may also depict interactions that illustrate system behavior.

Collaboration diagrams provide a view of the interactions or structural relationships between objects in the current model. This type of diagram emphasizes the relationship between objects whereas sequence diagrams emphasize the sequence of events. Collaboration diagrams contain objects, links, and messages. Use collaboration diagrams as the primary vehicle to describe interactions that express decisions about system behavior.

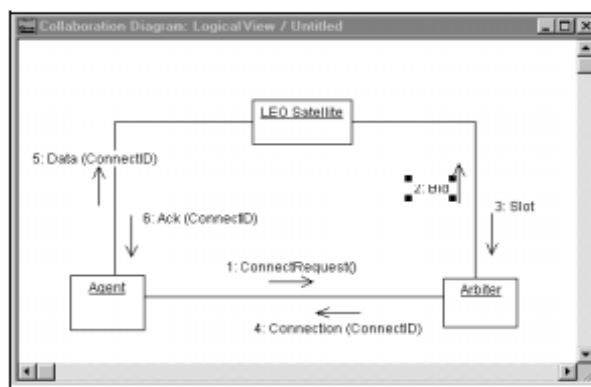


Figure 24: Collaboration Example

13.2.1 Collaboration Diagram Toolbox

The graphic below shows all the tools that can be placed on the collaboration diagram toolbox. The application window displays the following toolbox when the current window contains a collaboration diagram and you have selected **View > As Unified**.

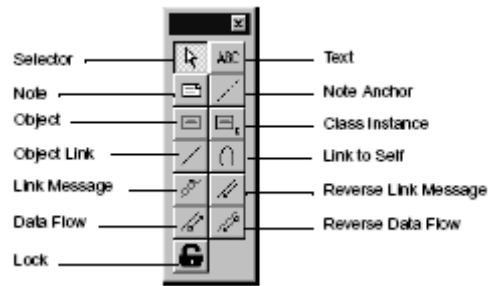


Figure 25: Collaboration Toolbox

13.3 Sequence Diagrams

A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence—what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. Sequence diagrams are normally associated with use cases.

This type of diagram is best used during early analysis phases in design because they are simple and easy to comprehend. A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

Sequence diagrams are closely related to collaboration diagrams and each are alternate representations of an interaction. A sequence diagram traces the execution of a scenario in time.

A sequence diagram illustrates object interactions arranged in a time sequence. These diagrams are typically associated with use cases. Sequence diagrams show you step-by-step what has to happen to accomplish something in the use case. This type of diagram emphasizes the sequence of events, whereas collaboration diagrams (an alternative view of the same information) emphasize the relationship. This type of diagram is best used early in the design or analysis phase because it is simple and easy to comprehend.

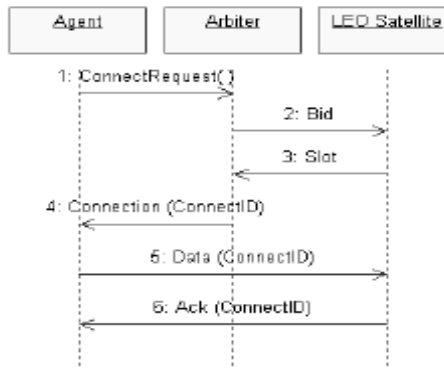


Figure 26: Sequence Diagram

13.3.1 Sequence Diagram Toolbox

The graphic below shows all the tools that can be placed on the sequence diagram toolbox. Refer to *Customizing the Toolbox* on page 14 for information on adding or deleting tools on a diagram toolbox.

The application window displays the following toolbox when the current window contains a sequence diagram and you have selected **View > As Unified**.

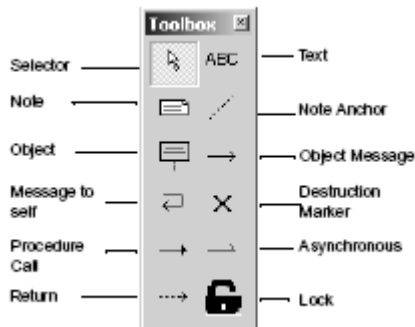


Figure 27: Sequence Toolbox

13.3.2 Object

One of the primary elements of a collaboration or sequence diagram is an object. An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance.

The object icon is similar to a class icon except that the name is underlined. If you use the same name for several object icons appearing in the same collaboration diagram, they are assumed to represent the same object; otherwise, each object icon represents a distinct object. Object icons appearing in different diagrams denote different objects, even if their names are identical. Objects can be named three different ways: object name, object name and class, or just by the class name itself.

13.3.3 Multiple Objects

If you have multiple objects that are instances of the same class, you can modify the object icon by selecting the **Multiple Instances** check box in the **Object Specification**. When you select this check box, the icon is changed from one object to three staggered objects.

13.3.4 Messages

A message icon represents the communication between objects, indicating that an action will follow. Each message icon represents a message passed between two objects, and indicates the direction a message is going. A message icon in a collaboration diagram can represent multiple messages. A message icon in a sequence diagram represents exactly one message. A message is the communication carried between two objects that triggers an event. A message carries information from the source focus of control to the destination focus of control. A message is represented on collaboration diagrams and sequence diagrams by a message icon which visually indicates its synchronization. The synchronization of a message can be modified through the message specification.

If all messages represented by a message icon do not have the same synchronization, the simple message icon is displayed. You can change the synchronization of the message by editing the message specification.

13.3.5 Links

Objects interact through their links to other objects. A link is an instance of an association, analogous to an object being an instance of a class. A link should exist between two objects, including class utilities, only if there is a relationship between their corresponding classes. The existence of a relationship between two classes symbolizes a path of communication between instances of the classes: one object may send messages to another. Links can support multiple messages in either direction. If a message is deleted, the link remains intact.

13.3.6 Sequence Numbering

Sequence numbering allows you to clearly see how messages interact and relate to one another. Numbering messages can be done two ways on sequence diagrams: top level numbering (a 1, 2, 3 pattern) or hierarchical numbering (a 1.1, 1.1.2, 1.1.3 pattern). Only top level numbering is available on collaboration diagrams. However, if you create a collaboration diagram from a sequence diagram with hierarchical numbering, the hierarchical numbering is retained.

- **Top-Level Numbering**

Top-level numbering gives each message or message to self a single number. There are no number subsets. Top-level numbering is useful in small sequence diagrams with few objects and messages.

- **Hierarchical Numbering**

Hierarchical numbering bases all messages on a dependent message. For example, you could have messages numbered 1, 1.1, 1.2, 1.2.1, where message number 1 is an independent message. All other message numbers numbered 1.x and beyond are dependent on message 1. If you remove independent message 1 from the diagram, all dependent messages will be removed.

To display hierarchical numbering:

- 1 Click **Tools > Options**.
- 2 Click the **Diagram** tab.
- 3 Select the **Sequence Numbering** check box.
- 4 Select the **Hierarchical Messages** check box.

13.3.7 Scripts

Scripts are used to enhance messages on sequence diagrams; they are text fields that attach to messages.

To create and attach a script:

- 1 Click the message icon and drag it between two objects.
- 2 Create text by either:
 - Using the **ABC** icon.
 - Clicking **Tools > Create >Text**.
- 3 Select one or more labels. Press the CTRL or SHIFT key to enable multiple selections.
- 4 Select one message.
- 5 Click **Edit > Attach Script** to attach the script to the message.

13.3.8 Focus of Control

Focus of Control (FOC) is an advanced notational technique that enhances sequence diagrams. This technique shows the period of time during which an object is performing an action, either directly or through an underlying procedure.

FOC is portrayed through narrow rectangles that adorn lifelines (the vertical lines descending from each object). The length of an FOC indicates the amount of time it takes for a message to be performed. When you move a message vertically, each dependent message will move vertically as well. Also, you can move an FOC vertically off the source FOC to make it detached and independent.

14. Component Diagrams

14.1 Overview

A component diagram shows the physical dependency relationships (mapping to a file system) between components—main programs, subprograms, packages, and tasks—and the arrangement of components into component packages.

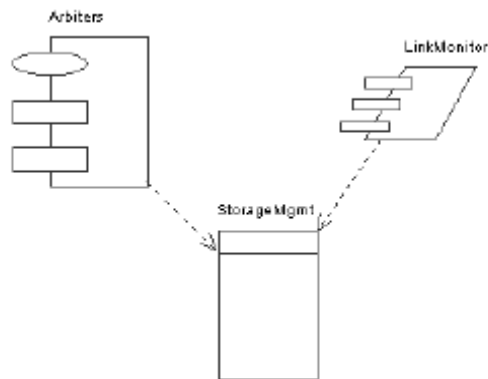


Figure 28: Component Diagram

Component diagrams are contained (owned) either at the top level of the model or by a package. This means the diagram will depict the components and packages in which the diagram is contained.

14.2 Creating and Displaying a Component Diagram

You can create or display the component diagram in one of three ways:

- Click **Browse > Component Diagram**.
- On the toolbar, click the component diagram icon.
- On the browser, double-click the component diagram icon.

14.3 Component Diagram Toolbox

The application window displays the following toolbox when the current window contains a component diagram and **View > As Unified** is selected.

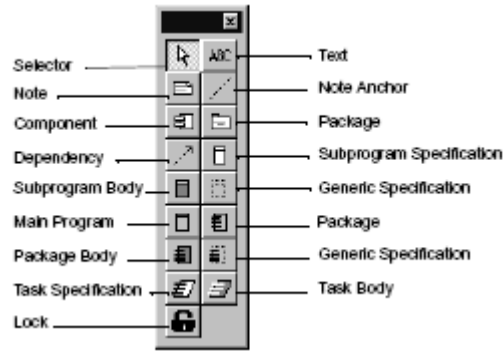


Figure 29: Component Toolbox

15. Deployment Diagrams

15.1 Overview

A deployment diagram shows processors, devices, and connections. Each model contains a single deployment diagram that shows the connections between processors and devices, and the allocation of its processes to processors.

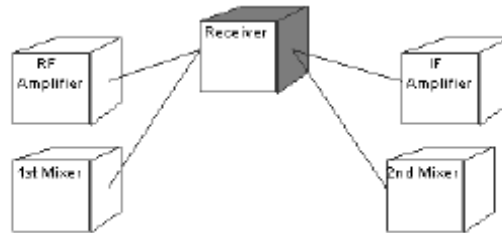


Figure 30: Deployment Example

15.2 Creating and Displaying a Deployment Diagram

You can create or display the deployment diagram in one of three ways:

- Click **Browse > Deployment Diagram**.
- On the toolbar, click the deployment diagram icon.
- In the browser, double-click the deployment diagram icon.

15.3 Deployment Diagram Toolbox

The application window displays the following toolbox when the current window contains a deployment diagram and you have selected **View > As Unified**:

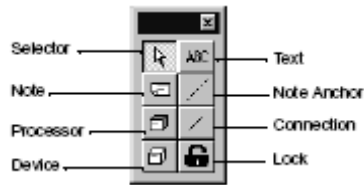


Figure 31: Deployment Toolbox

15.4 Processor Specification

A **Processor Specification** displays and modifies the properties and relationships of a processor in the current model. Some of the information on the specification can also be displayed inside icons representing the processor in a model's deployment diagram.

15.5 Processor

The owner of the process is shown here.

15.6 Priority

This field contains the relative priority of this process, if there is one. You can use this information with the scheduling type identified in the **Processor Specification** to schedule process execution.

15.7 Characteristics

Use the **Characteristics** field to specify a physical description of an element. For example, you can describe the kind and bandwidth of a connection; the manufacturer, model, memory, and disks of a machine; or the kind and size of a device. You can set this field only through the specification. This information is not displayed in the deployment diagram.

To update this field, click the **Characteristics** field and enter the information.

Processes Use this field to identify the processes assigned to this processor.

Processes denote either the root of a main program from a component diagram or the name of an active object from a collaboration diagram.

To create a process, right-click in the processes area and click **Insert** from the shortcut menu. A new process entry is created. To change the name or priority, click the item and type the changes. You can display a list of the processes by selecting the processor icon and clicking **Show Processes** from the shortcut menu.

15.8 Scheduling

The **Scheduling** field specifies the type of process scheduling used by the processor. Use these options to specify the appropriate scheduling.

Type	Description
Preemptive (default)	Higher-priority processes that are ready to execute can preempt lower-priority processes that are currently executing. Processes with equal priority are given a time slice in which to execute, allowing computation resources to be fairly distributed.
Non preemptive	The current process continues to execute until it relinquishes control.
Cyclic	Control passes from one process to another; each process is given a fixed amount of processing time.
Executive	An algorithm controls process scheduling.
Manual	Processes are scheduled by a user outside of the system.

Table 6: Scheduling Options

16. Case Study:

Student Registration System

16.1 Problem Statement

16.1.1 Case Study Background

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.

16.1.2 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.

16.1.3 Project Summary

This system will have a short inception phase during which prototyping is used to select the database. The use case diagram is started in the inception phase and matured in the elaboration phase. By the end of the elaboration phase, an architectural iteration is complete. The system is evolved in the construction phase in two iterations. The process components of requirements analysis, design, implementation and test are used in all phases of the project lifecycle.

16.2 The Inception Phase

16.2.1 Business Goals and Needs

The first question to address is the need for a new registration system. Does the University have the resources needed to design and implement the new system? In addition to the assessment of need for the system, the risks posed by the new system are elaborated. In the case of an on-line registration system, one of the major risks is the ability to store the information in a manner that is easily and quickly accessible by all.

For the purposes of this case study it was decided that the new system should be built. Prototypes were completed to address the database risks.

16.2.2 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.

16.2.3 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.

16.2.4 Use Case Diagram in Rational Rose

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 in Figure.

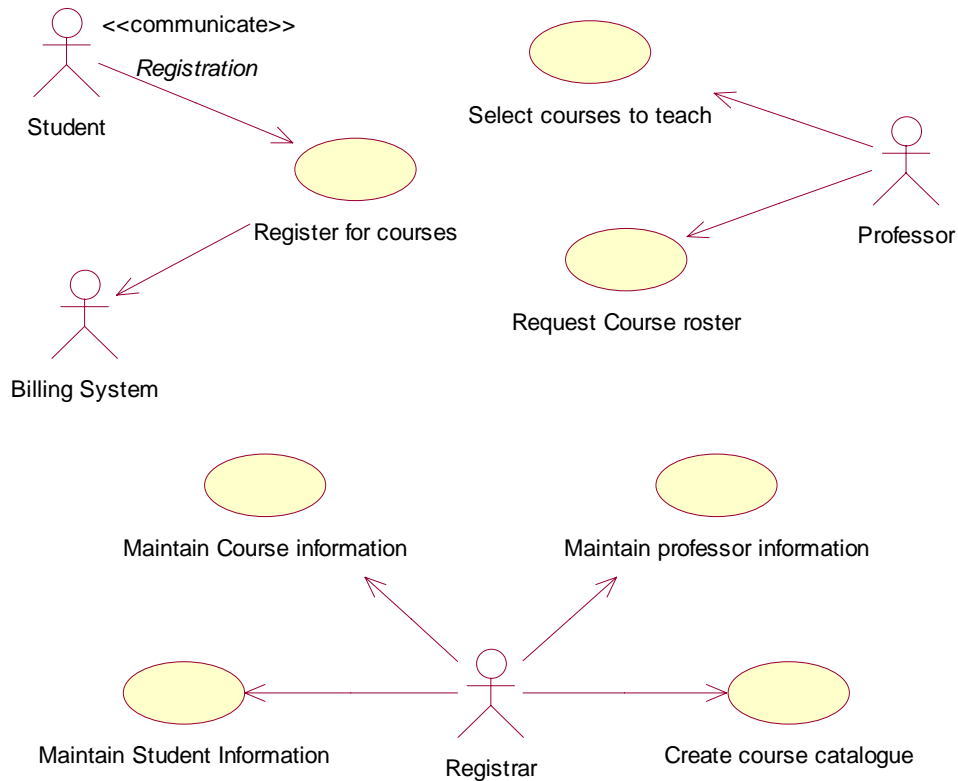


Figure 32: Use Case Diagram

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

- Register for courses
 - The use case is started by the student. 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
 - This use case is started by the professor. It provides the capability to request a printed list of all students assigned to a specified course offering.
- Select courses to teach
 - This use case is started by the professor. It provides the capability to select, review, modify, and delete a list of courses to teach for a specified semester.

- Maintain professor information
 - This use case is started by the registrar. It provides the capability to create, review, modify, and delete professor information.
- Maintain student information
 - This use case is started by the registrar. It provides the capability to create, review, modify, and delete student information.
- Maintain curriculum
 - This use case is started by the registrar. It provides the capability to create, review, modify, and delete a list of course offerings for a given semester.
- Generate catalogue
 - This use case is started by the registrar. It provides the capability to generate a catalogue containing a list of course offerings for a specified semester.

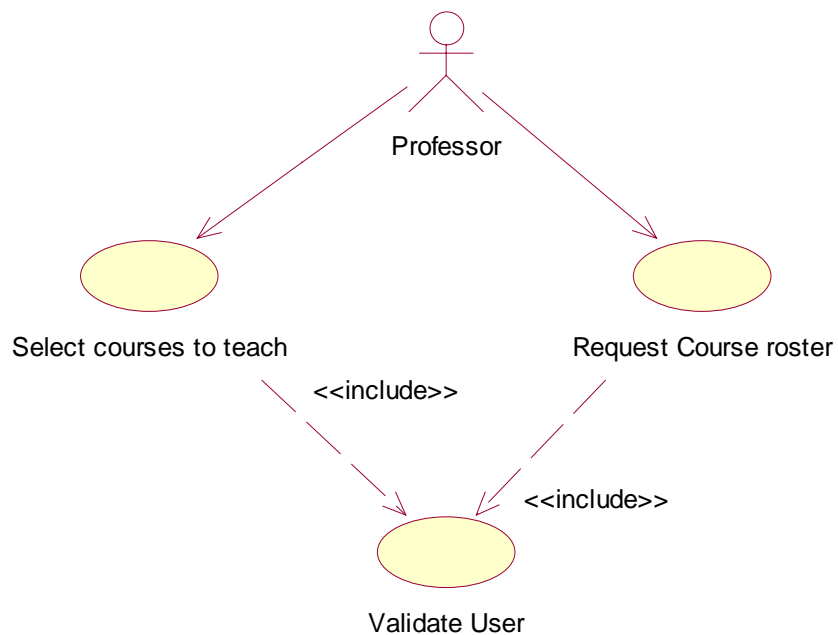


Figure 33: Use Case Dependency

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

In Rose, the flow of events is entered via a link to an external document. The flow of events for the Register for Courses use case is shown below.

16.2.5 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: Register for Courses Use Case

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.

16.2.5.1 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:

1. Checks that prerequisites are satisfied for the requested course.
2. 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.

16.2.5.2 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, course offering number, days of the week, time, location, and number of credit hours.

16.2.5.3 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.

16.2.5.4 Change Schedule - 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:

1. Verifies that the maximum course load for the student has not been exceeded.
2. Checks that prerequisites are satisfied for the requested course.
3. Adds the student to the course offering if the course offering is open.

16.2.6 Flow of Events: Select Courses to Teach

16.2.6.1 Brief Description

This USE Case begins when the professor logs onto the registration system and enters his/her password. The system verifies that the password is valid (if password is invalid, Alternate flow is executed) and prompts the professor to select the current semester or a future semester (if an invalid semester is entered, Alternate flow executed). The professor enters the desired activity: ADD, DELETE, REVIEW, PRINT or QUIT.

16.2.6.2 Basic Flow

If the activity is ADD, the system displays the course screen containing a field for a course name and number. The professor enters the name and number. The professor enters the name and number of a course (if an invalid name/number is entered, alternate flow executed). The system displays the course offerings for the entered course (if the course name cannot be

displayed, alternate flow executed). The professor selects a course offering (if link cannot be created, alternate flow executed). The system links the professor to the selected course offering. The use case then begins again.

If the activity selected is DELETE, the system displays the course offering screen containing a field for a course offering name and number. The professor enters the name and number of a course offering (if invalid number/name combination is entered, alternate flow executed). The system removes the link to the professor (if the link cannot be removed, alternate flow is executed). The Use Case then begins again.

If the activity selected is REVIEW, the system retrieves (if the course information cannot be retrieved, alternate flow executed) and displays the following information for all course offering for which the professor is assigned: course name, course number, course offering number, days of the week, time and location. When the professor indicates that he or she is through reviewing, the Use Case begins again.

If the activity selected is PRINT, the system prints the professor's schedule (if the schedule cannot be printed, alternate flow executed). The Use Case begins again.

If the activity is QUIT, the Use Case ends.

16.2.6.3 Alternate Flow

Invalid Password

An invalid password is entered. The user can re-enter a password or terminate the Use Case.

Invalid Semester

The system informs the user that the semester is invalid. The user can re-enter the semester or terminate the Use Case.

Invalid Course Name/Number

The system informs the user that the course name/number is invalid. The user can re enter a valid name/number combination or terminate the Use Case.

Course Offering Cannot be Displayed

The user is informed that this option is not available at this current time.
The Use Case begins again.

Cannot Create Link Between Professor and Course Offering

The information is saved and the system will create the link at a later time.
The Use Case begins again.

Link Between Professor and Course Offering Cannot be Removed

The information is saved and the system will remove the link at a later time.
The Use Case begins again.

Schedule Information Cannot be Retrieved

The user is informed that this is not available at the current time. The Use Case begins again.

Schedule cannot be printed

The user is informed that this option is not available at the current time. The use case begins again.

16.2.7 Activity Diagram

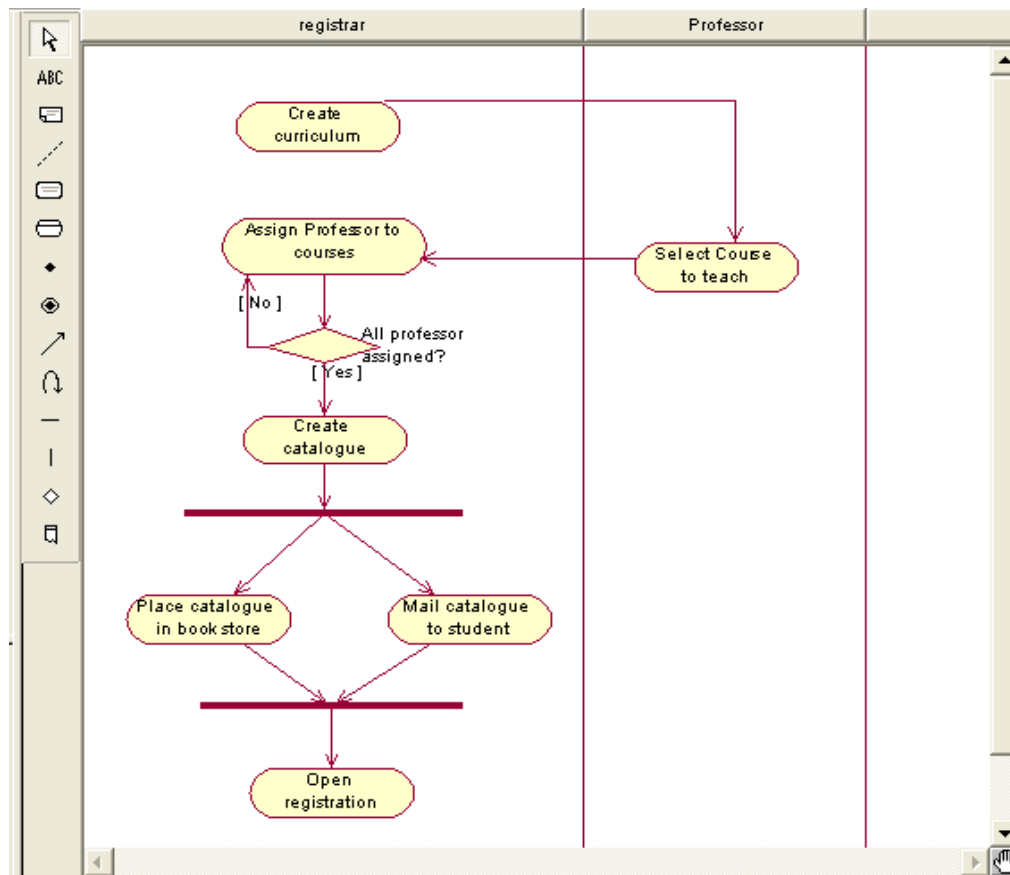


Figure 34: Activity Diagram For Create Catalogue

16.3 The Elaboration Phase

During Elaboration, some of the most important and critical use cases are implemented. During this phase, the focus is good class structure and architecture.

16.3.1 Development of Scenarios

Each use case is a web of scenarios. Scenarios are documented using Sequence Diagrams. Objects are represented as vertical lines and messages between objects are shown as directed horizontal lines. Sequence diagrams are drawn in the Use Case View of the tool. The Sequence Diagram for the Add a Course scenario is shown in Figure.

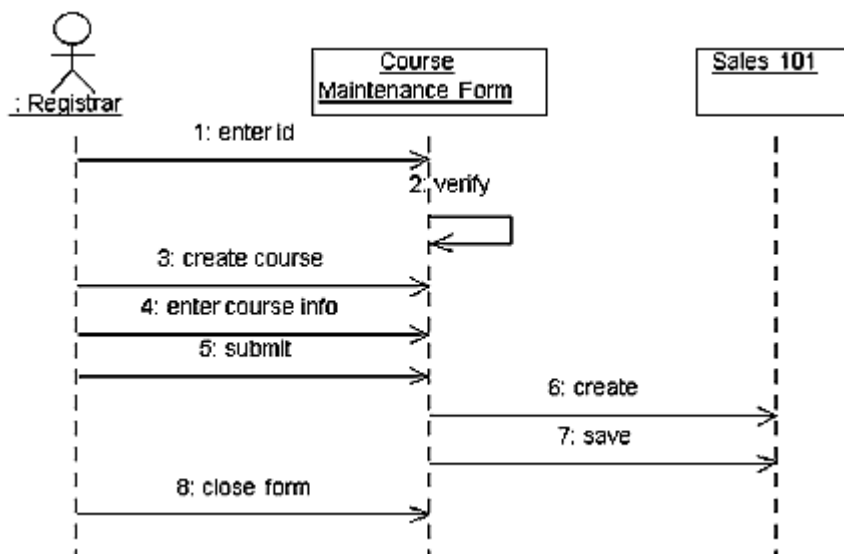


Figure 35: Sequence Diagram for the Add a Course Scenario

16.3.2 Creating "Real World" or "Business" Classes

Objects are discovered by examining the use cases and scenarios and grouped into classes. Each class should have a definition which states the purpose of the class. Packages are created to hold logical groups of classes. Classes and packages are drawn in the Logical View of the tool. The following packages and classes have been created for the registration system:

- 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, tenureStatus).
- UniversityArtifacts
 - Course--General information about selections for a semester (for example, name, description, creditHours).
 - CourseOffering--Specific information about selections for a semester (for example, daysOffered, timeOfDay, location).
 - StudentSchedule--Output report containing the list of registered course offerings generated when a student registers for a course.
 - CourseRoster--Output report containing the list of registered students for a specific course offering generated for a professor.
- Interfaces
 - RegistrationForm--Form which provides the capability for a student to select registration options.
 - Add/DropForm--Form which provides the capability for a student to modify a course schedule.
 - CourseSelectionForm--Form which provides the capability for a professor to add/drop courses to teach.
 - StudentMaintenanceForm--Form which provides the capability for the registrar to add/delete/modify student information.
 - ProfessorMaintenanceForm--Form which provides the capability for the registrar to add/delete/modify professor information.
 - CourseMaintenanceForm--Form which provides the capability for the registrar to add/delete/modify course and course offering information.

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. Class diagrams are contained in the Logical View of the tool.

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 Figures through. An updated sequence diagram is shown in Figure.

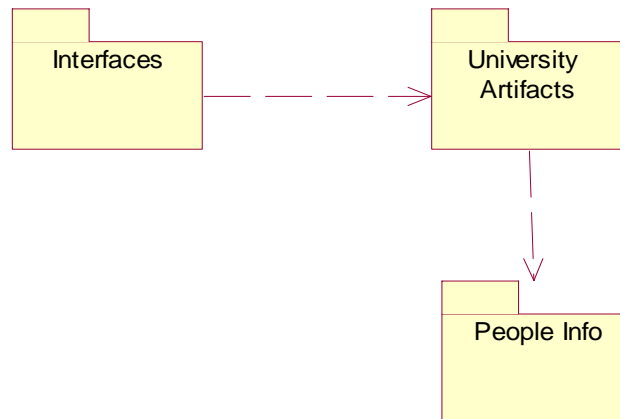


Figure 36: Main Class Diagram

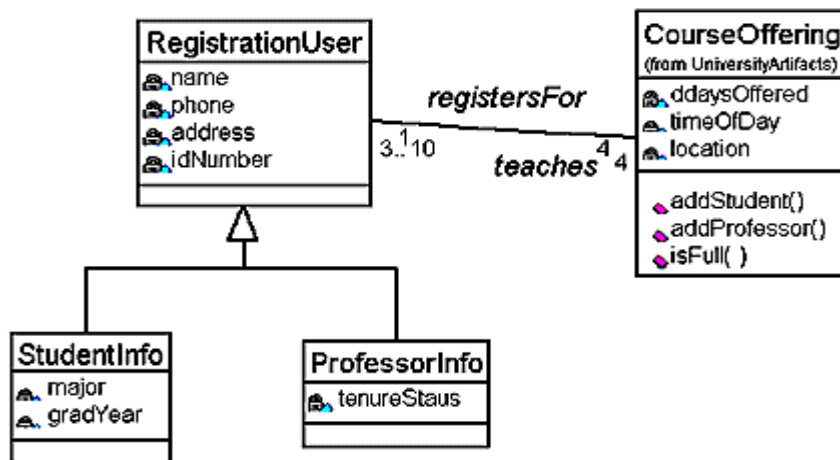


Figure 37: Main Class Diagram for the People Package

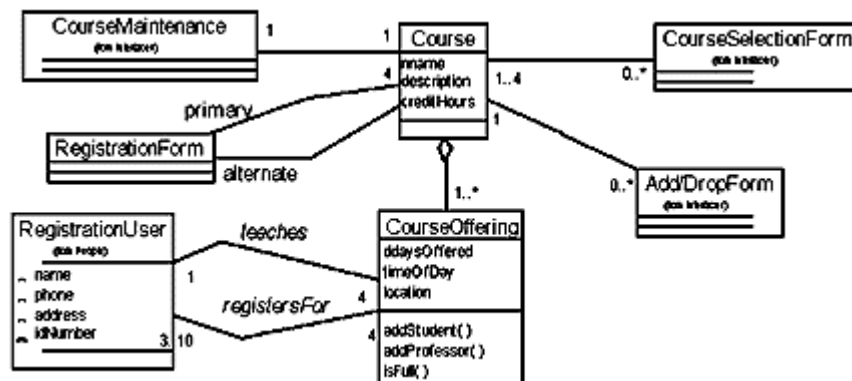


Figure 38: Main Class Diagram for the University Artifacts Package

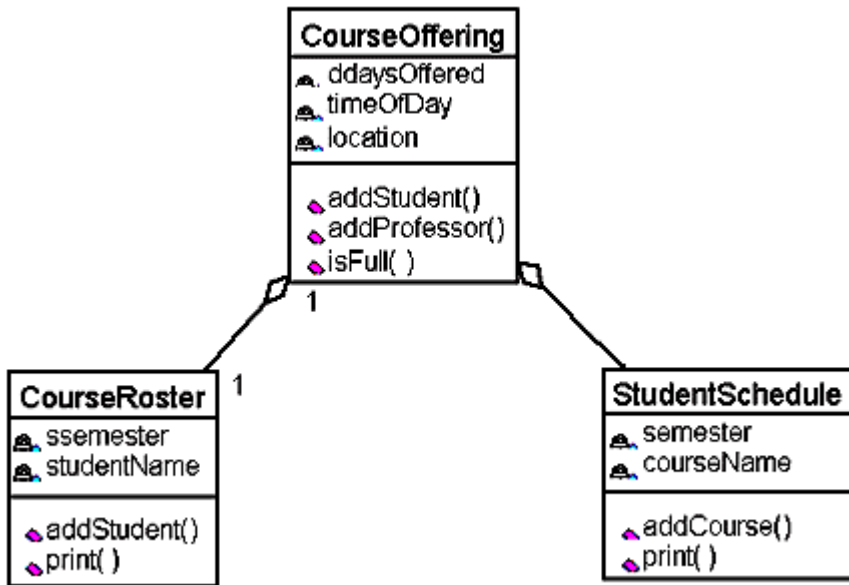


Figure 39: Course Reporting Class Diagram in the University Artifacts Package

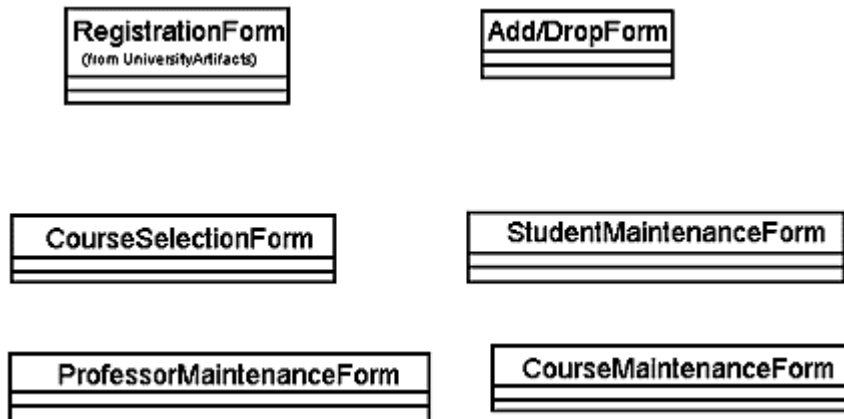


Figure 40: Main Class Diagram for the Interfaces Package

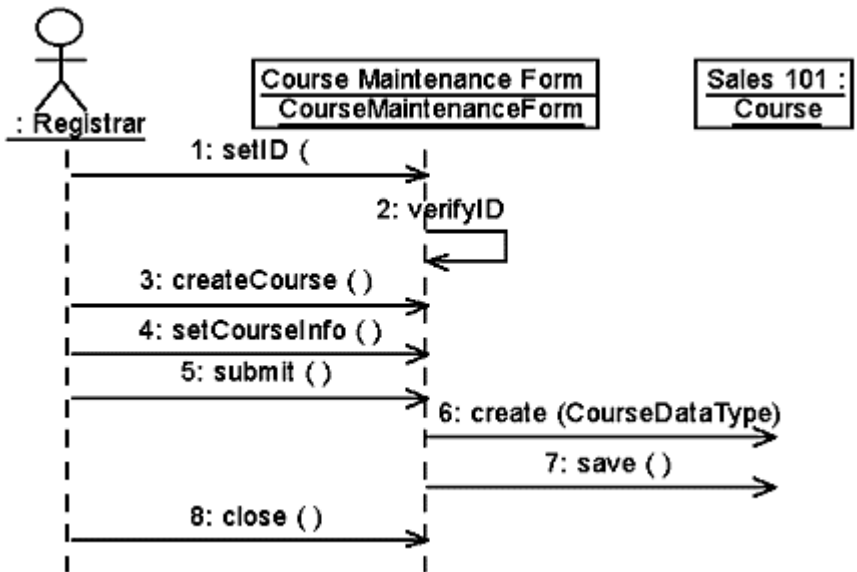


Figure 41: Updated Sequence Diagram

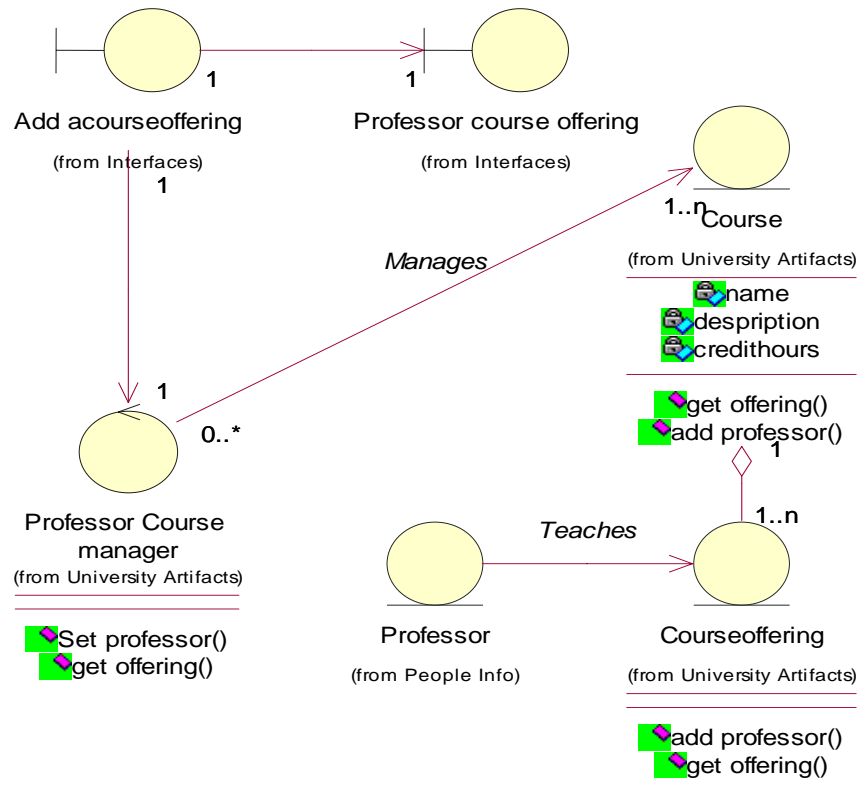


Figure 42: Class Diagram with Class types

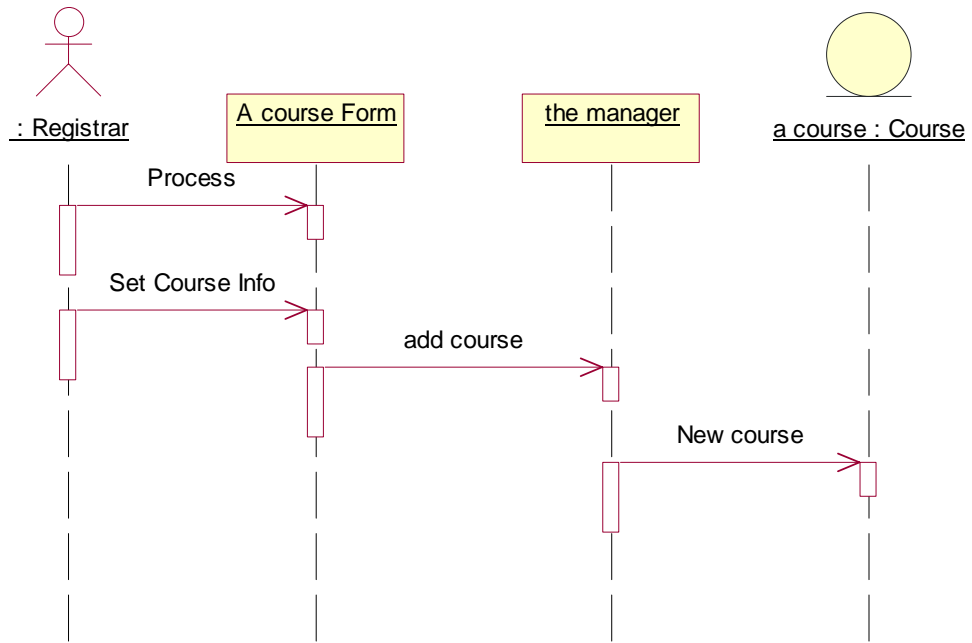


Figure 43: Sequence Diagram for Registrar

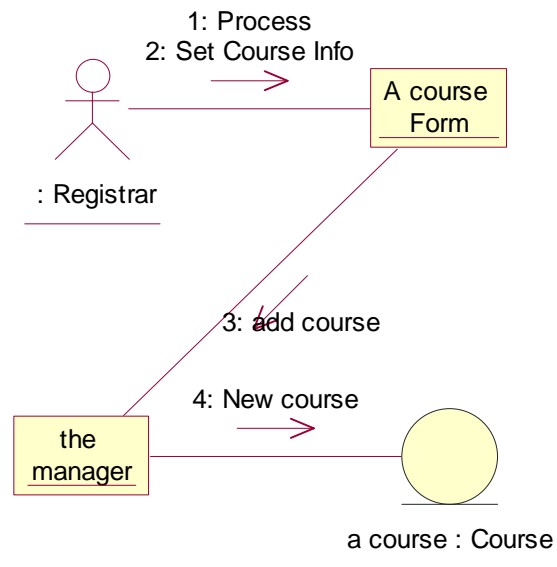


Figure 44: Collaboration Diagram for Registrar

16.3.3 Software Architecture

As the elaboration phase of development continues, decisions concerning the architectural framework for the project are made. Scenarios are updated to show the interaction of the real world objects with the objects representing the architectural decisions. Packages and classes that carry out the architectural functionality are added to the logical view.

In the Course Registration system, the following architectural decisions were made:

- Containers and GUI classes to be used are in the MFC class library.
- A commercial relational database was chosen and classes to communicate with the database were created.
- A set of error classes were created to facilitate common error handling strategies.

The updated Main Class Diagram and an updated Sequence Diagram are shown in Figures.

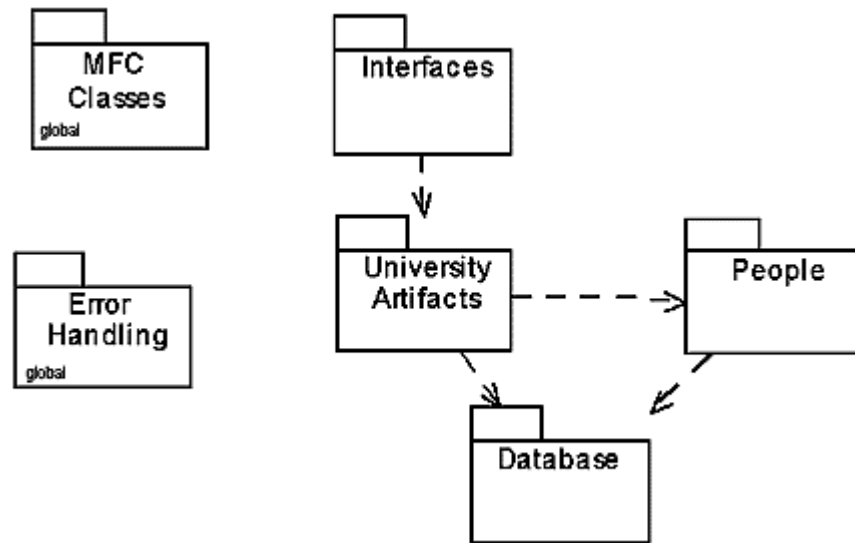


Figure 45: Main Class Diagram

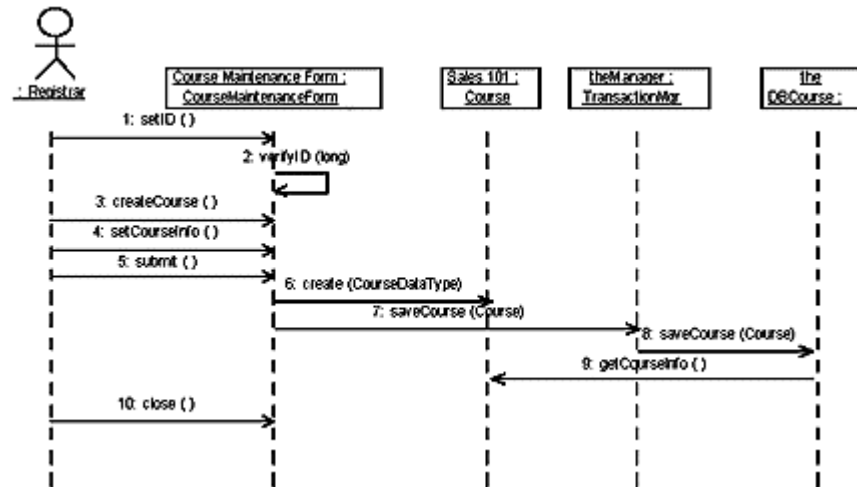


Figure 46: Updated Sequence Diagram

The next step is to implement a set of scenarios that address the major architectural issues. This is done to ensure early feedback and identification of problems. For this problem, the Maintain Curriculum Use Case was implemented since it addressed the major risk of this system--the database risk.

16.3.4 Iteration Planning

Another activity in the elaboration phase is the creation of the iteration plan. The goal of an iteration is to reduce risk in the system while incrementally building the final product. Use cases and scenarios are examined and prioritized to create the initial project plan. As each iteration is completed, risks are re-evaluated and the project plan is updated as needed.

For the Course Registration system the iteration plan is:

- Iteration 1
 - Maintain curriculum.
- Iteration 2
 - Maintain student info.
 - Maintain professor info.
 - Select courses to teach.
 - Generate catalogue.
- Iteration 3
 - Register for courses.
 - Request class roster.

16.4 The Construction Phase

16.4.1 Construction Activities

During Construction, all remaining scenarios will be specified and implemented. At this time, many of the secondary scenarios are addressed.

16.4.2 Building an Iteration

This case study concentrates on the "Add a Course" scenario which is shown in Figure. During this phase of development, the classes that participate in the iteration are designed and implemented. Class diagrams are created to show the focus of the iteration.

For the Course Registration problem, the following design decisions are made:

- Controller class—CurriculumManager added. This class knows the business rules associated with the management of a curriculum.
- Scenario diagrams are updated to show new interactions with the CurriculumManager.
 - Some interactions between objects are deleted due to the addition of the controller.
- Data types and signatures are provided for all attributes and operations.
- Association navigation is designed.
- Associations are changed into dependency relationships where appropriate.

An updated Sequence diagram showing the interaction with the added controller class is shown in Figure.

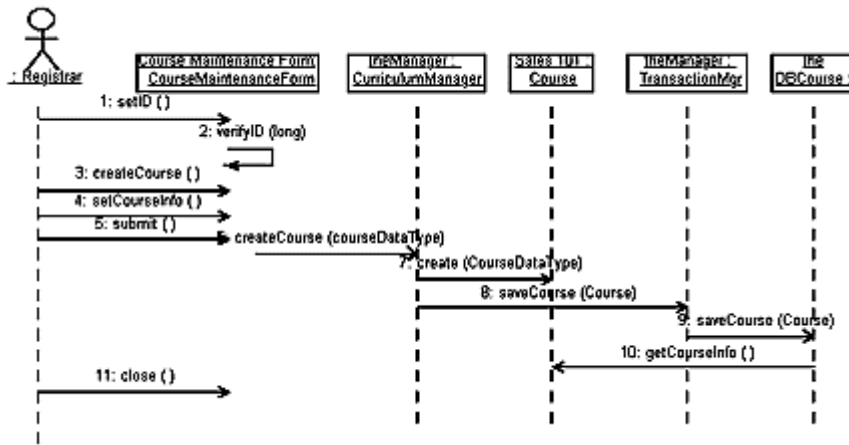


Figure 47: Updated Sequence Diagram

16.4.3 Add a Course

A package called Iteration 1 is added to the logical view of the model. Class diagrams showing the classes in the iteration are added to the package. A class diagram showing the design decisions made for the "Add a Course" scenario is shown in Figure.

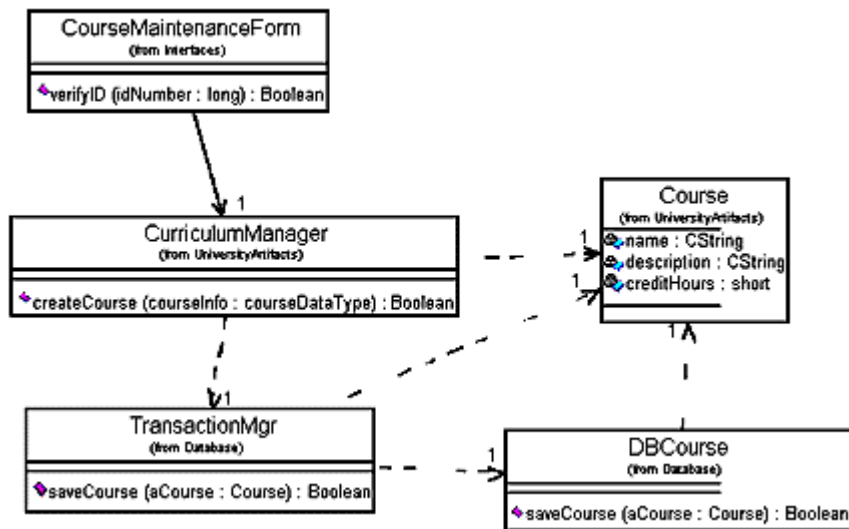


Figure 48: Class Diagram "Add a Course"

The code for the iteration is completed and the iteration is tested and documented. The completed iteration is integrated with any previous iteration.

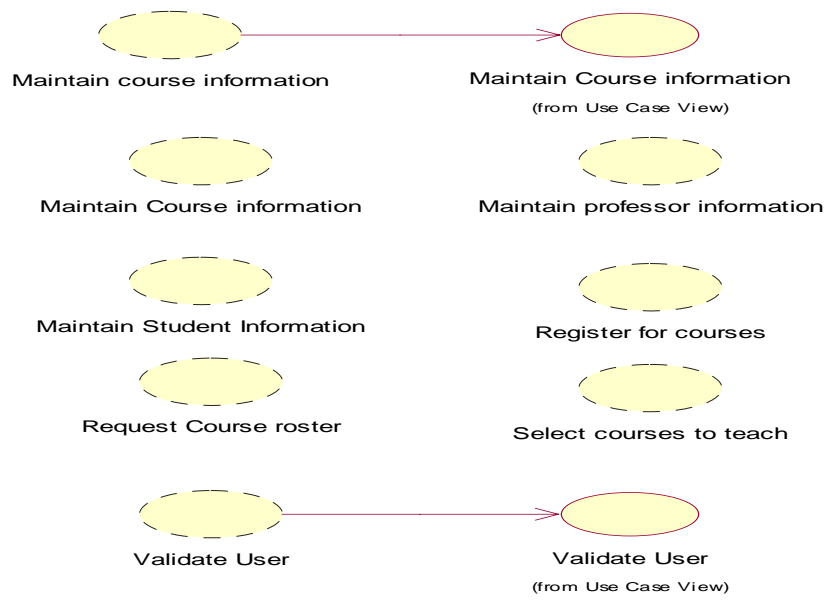


Figure 49: Realization Diagram

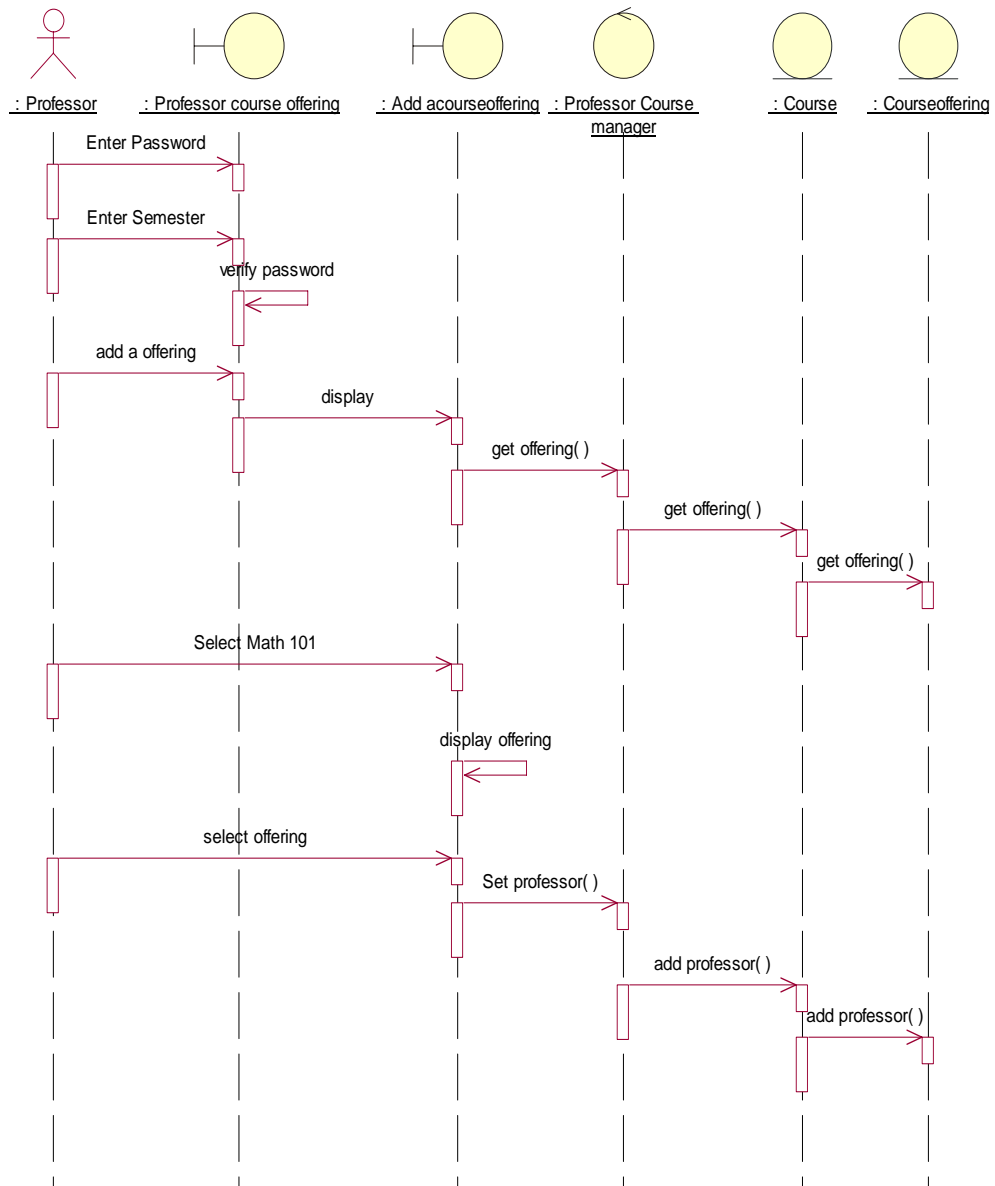


Figure 50: Sequence Diagram for Add a course offering

16.4.4 State Chart Diagram

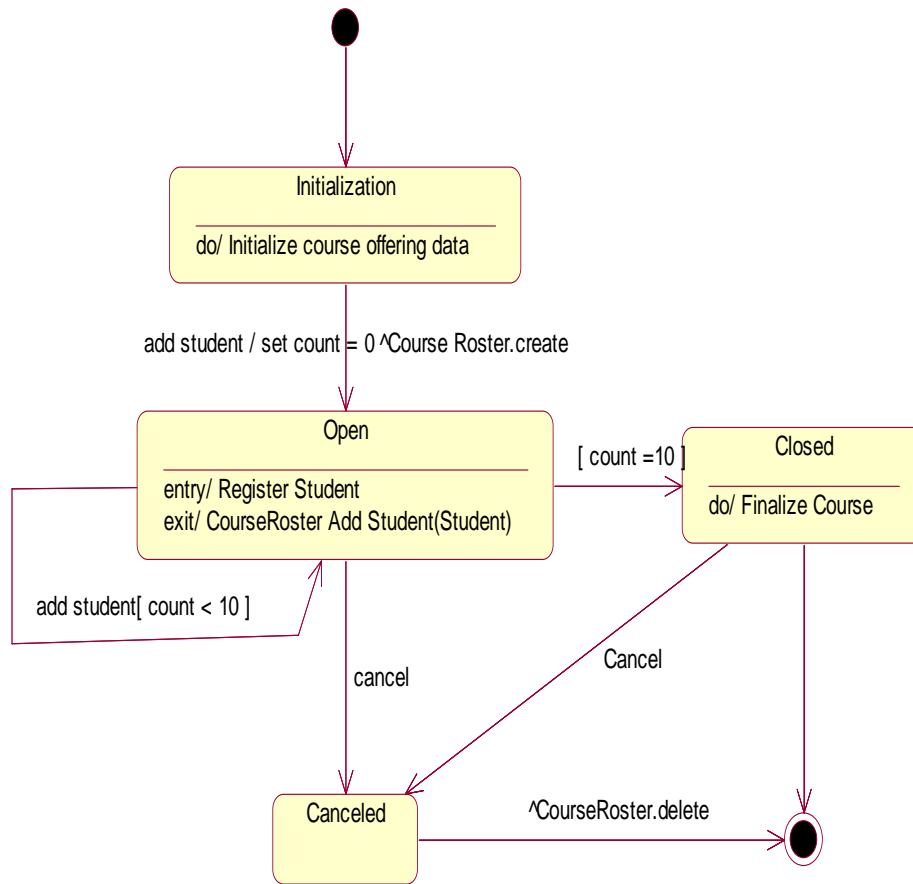


Figure 51: Course Offering States

16.4.5 Main Component Diagram

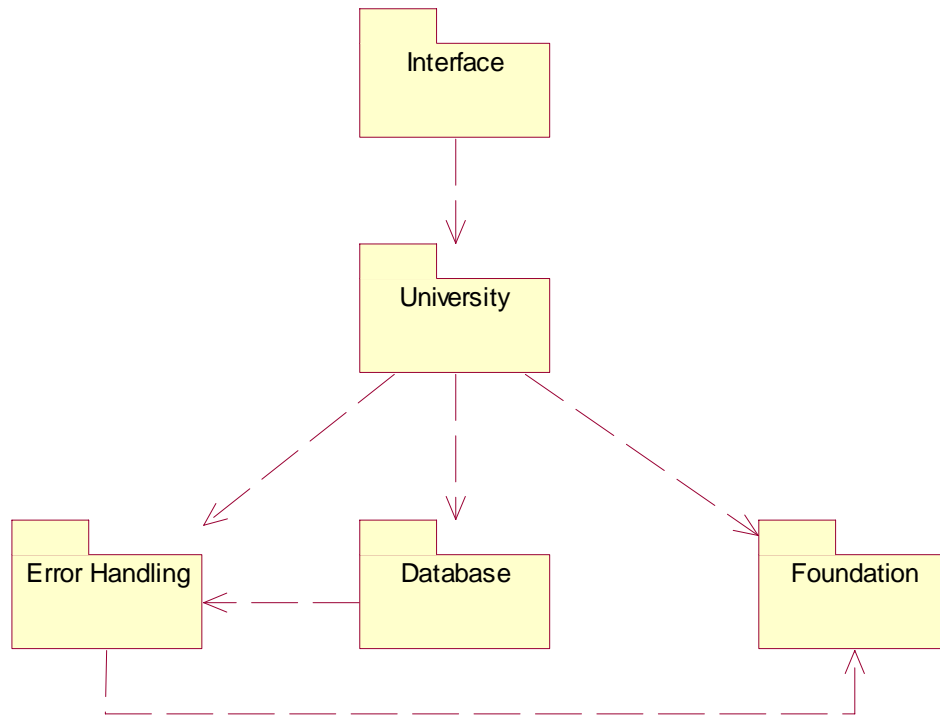


Figure 52: Main Component Diagram

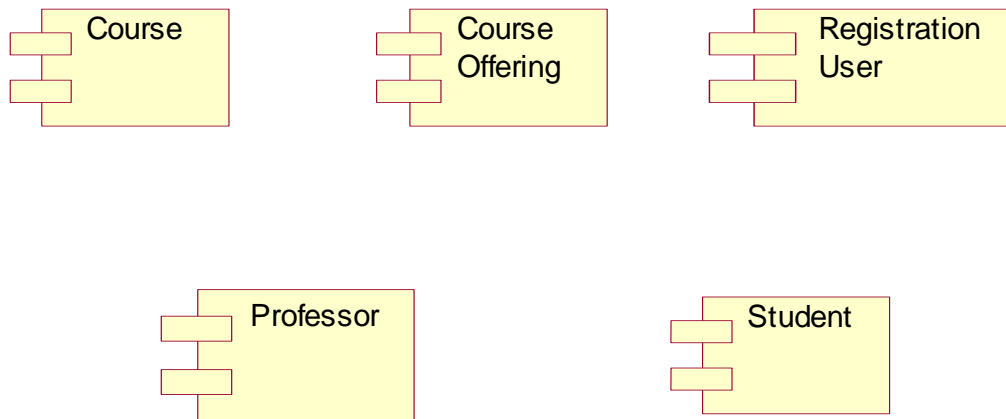


Figure 53: University Main Component Diagram

16.5 The Transition Phase

The system was successfully transitioned to the University community in two releases--beta and the final system. During the beta period, bugs were discovered, reported and fixed by the development staff. After using the beta version of the system, professors added the requirement to view a class roster on-line. This requirement was successfully implemented and available in the final release of the system. Students and professors were pleased with the time savings provided by the paperless system.

Due to the success of the Registration System it was decided that another version of the system should be developed to provide an on-line catalogue of course offerings. Budgets and staff were approved and the process began again.

16.5.1 Deployment View

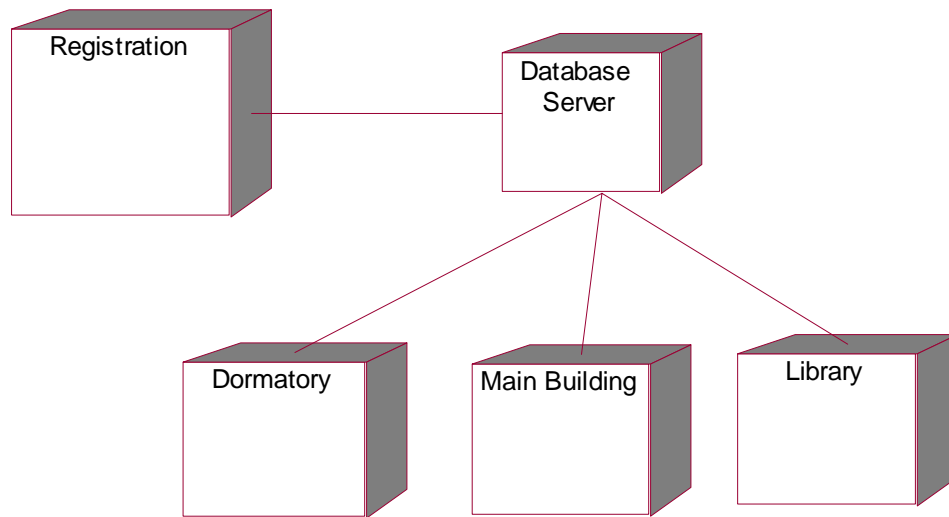


Figure 54: Main Deployment Diagram

17. Case Study: Online Auction System

17.1 Problem Statement

As the head of Information Technology at Crimson Systems Ltd. you are tasked with developing a Real Time Auction System.

The new system will be a web-based application available to everyone around the world. The bidding process will start with a registration process to collect user details and transmission information from the participants. The user will have the choice to enter for bidding and/or for selling items. There will be a cart which will display all available items along with their current price, condition of item and the end date of bidding for that particular item. The favorites / Bestsellers items will also be displayed.

If the user wants to sell an item then he/she has to give the details of the item for bidding and if the user wants to start bidding then he/she must check out the item details along with the details of the item's seller before making a bid. Bidding process will be conducted in rounds. In each round there will be a timing constraint in which a client should make a bid. At the end of the bid, the server will determine the highest bid received and broadcasts this to all the clients. This process will continue till no client sends a new bid or the highest bid remains constant for three consecutive rounds. The final result will then, be broadcasted by the server to all clients and the bidding process will be terminated.

If the user makes the highest bid then he/she will be informed via email and he/she will be required to give the order details. If the user is out bided then also he/she is informed via email about the winning bid and runner ups.

The system aims to provide reliable real time open-bid auction i.e. all customers should come to know of a bid made by someone else in a minimum amount of time. Moreover, the auctioning protocol used will be reliable which means that it will have a bounded and predictable communication delay.

In addition to real-time concerns associated with auctions, there are also privacy concerns. This makes sealed bid auctions necessary. So, no auction bid should be revealed except for the winning bid and the runner ups, no winner should be able to repudiate his bid and the auction should be carried out in real-time.

One of the most important issues related to real time auction is the payment method. This will be secure, reliable, scalable, anonymous, flexible, convertible, integrated and easy to use. The methods used will be Cash on Delivery, Cheque, and ATM Drop Box.

The system will be completely distributed; Scalable and synchronization of all the components will be maintained.

17.2 Real Time Auction System Glossary

1. Introduction

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

2. Definitions

The glossary contains the working definitions for the key concepts in the Real Time Auction System.

2.1 Real Time System

It is a system having well-defined, fixed time constraints.

2.2 Bidding

Propose a payment; as at sales or auctions

2.3 Registration Process

The process by which each user enters his/her details before he/she starts selling/bidding.

2.4 Cart

It is like a wagon containing all items to be auctioned

2.5 Order Details

It will contain the details of the item for which the highest bid was made by the user along with the delivery charges and payment methods .

2.6 Protocol

It is a set of rules that govern the format and transmission of data across any two corresponding layers

2.7 Condition of item

It tells whether the item is first hand or second hand.

2.8 Distributed System

It is distributed computing system that uses distributed operating system.

2.9 Scalable System

This is the one in which addition and deletion of users has no effect on the functionality.

2.10 Secure Payment

To preserve integrity of payment data.

2.11 Reliable Payment

So that the infrastructure can withstand denial of service attacks and network failures.

2.12 Scalable Payment

The payment method will be able to handle increase /decrease in number of users without deterioration of performance.

2.13 Anonymous Payment

The identity of the users to the transaction will be protected and it will not be possible to monitor an individual's spending patterns or to find out his source of income.

2.14 Flexible Payment

It will be possible to incorporate other payment methods.

2.15 Convertibility in Payment

It will be possible to convert one electronic currency to another.

2.16 Integrated Payment

The initial set-up cost should not be high

17.3 System Requirement Specifications

1 Introduction

1.1 Purpose

This document deals with the detailed specification based on complete analysis, study, environment, background, manual interactions of the system and complete analysis of the problem statement. It provides an overview of the project REAL TIME AUCTION SYSTEM. The system developed would be web based.

1.2 Scope

This document is prepared after understanding the exact requirements of the Project. It documents all the functions, performance and interfacing requirements for the software. Thus it describes the 'what' of the system

1.3 Definitions, Acronyms, Abbreviations

Definitions

Acronyms

Name

Buyer

Zip Code

Seller

Logout

Not applicable

Acronym

Customer, Consumer

Postal Code, Pin Code

Retailer

Signout

Abbreviations

Name

Date of Birth

Payment Preferences

Hot Selling

Ending Soon

Second Hand Goods

Paisa Auction

1 Paisa Auction

View Cart

Seller View

Telephone Number

Abbreviation

DOB, Dob

PayPref

HotS

EndS

SHGoods

PA

OnePA

BView

SView

Tel

STD Code	STD
Contact Time	CTime
Extra Charges	Extra
Current Bid History	CBHist
Warranty Options	Warr
Current Price	CP
Indicative MP	MP
Starting Bid	SBid
Your Maximum Bid	YMBid
Confirm Password	CPass
Confirm ID	CID
Update Profile	Pupdate
Update Password	Passup
Update email id	IDup
Boli Password	PBoli
Boli ID	IDBoli
Balance	Bal
Memory	Mem
List of Items	Litem
Select Item	Sitem
Bid Now	BID
Location	Loc
Member	Memb

1.4 References

Crimson Systems Ltd. is one of the leading companies in web-based applications. The real time auction system is one of the emerging services that would be provided by Crimson Systems Ltd. The project is being developed by the Research Department. The Project has been named “BOL BOLI BOL”.

1.5 Overview

The Real Time Auction System provides its users the facility for bidding for items along with the option to enter their items for auction. The whole process is aimed at being optimizing, to ensure proper co-ordination. The results of each bid are mailed to the participants of the particular bid.

2 Overall Descriptions

2.1 Product Perspective

2.2 Product Functions

1 Join Now	2 UserID	3 Password
4 Login	5 Forgot Password	6 OK
7 Home	8 Name	9 Email ID
10 Confirm ID	11 Boli ID	12 Boli Password
13 Date of Birth	14 Accept terms and Submit	15 Cancel
16 Help	17 Start Selling	18 View Cart
19 Buyer View	20 Seller View	21 Update Profile
22 Update Password	23 Update email id	24 Logout
25 Favorites	26 Hot Selling	27 Ending Soon
28 Second Hand Goods Auction	29 Paisa Auction	30 1 Paisa
31 List of Items	32 Payment preferences	33 Extra Charges
34 Current Bid History	35 Bid Win	36 Bid Lost
37 Balance	38 Address	39 Country
40 City	41 Zip Code	42 Contact Time
42 STD Code	43 Telephone Number	44 Select Item
46 Condition	47 Warranty Options	48 Items
49 Picture	50 Type	51 Current Price
52 Ends	53 Bid Now	54 Starting Bid
55 Indicative MP Maximum	56 Time Left	57 Your Bid
57 Quantity (Seller)	58 Item Description	59 ID
61 Location (Seller)	62 Member Since (Seller)	63 Place a Bid
64 Model	65 Features	66 Memory
67 List Item	68 Go back to Cart	

2.3 User Characteristics

The online version of THE Real Time Auction System will be launched in the market on 1st October 2005. Along with the software the package will contain user and system manuals.

User Manual : To guide the user regarding the use of the software ,what to do in case of troubleshooting

User Manual for Time Auction System in Crimson Systems Ltd.

System Manual : Which will tell how the package functions, what should be the inputs, what would be the outputs, as well as the complete documentation.

System Manual for Time Auction System in Crimson Systems Ltd.

2.4 Constraints

Minimum Hardware Specification For Server

40 GB Hard disk
128 MB RAM
52X CD-ROM
Pentium-IV Processor
(Minimum 32 Bit Processor)
MotherBoard

Implementation Language ASP

OS Used
Windows 2000 Advanced Server(for server)
Windows 98 and onwards(for client)

Security Considerations
All the modules must be properly integrated together. Loopholes

in the software must be avoided to prevent any possible security breach. Necessary security parameters must be considered and integrated .

2.5 Assumptions and Dependencies

Assumptions

- No time Delay
- Rollback and error recovery are possible in every case
- No loopholes are present in the software developed
- Fully secure
- User checks both item & seller details before bidding
- The system uses sealed bid auctions
- Auctions are carried out in real time
- The server itself determines the highest bid during successive rounds and broadcasts the results accordingly.

- No user can access other users account or details

Dependencies

- Bid results depend on date and time of bidding
- The major factor on which the whole application depends is Communication and Transmission time
- Buyer as well as Seller profile contains details which are dependant on the bids made and transaction completed

3 Specific Requirements

3.1 External Interfaces

Total 21 Screens and 68 Items

No.1

(i)Name of item	Join Now
(ii)Description of purpose is	For making new users this link pressed
(iii)Source of input	NA
(iv)Valid range & accuracy	NA
(v)Units of measurement	NA
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs this	User profile created by function

(viii)Screen format	2 input fields,1 push button,3 links
(ix)Data format	NA
(x)Window format resolution	Default Window having 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.2

(i)Name of item	UserID
(ii)Description of purpose an	Name of user to identify him as authenticated user
(iii)Source of input	RTAS
(iv)Valid range & accuracy the first	20 Characters(max) and must be an alphabet
(v)Units of measurement underscore	Alphabets ,numbers &
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs access	Allows user to log into & the auction system
(viii)Screen format	2 input fields,1 push button,3 links
(ix)Data format input	Data displayed as the values from keyboard
(x)Window format resolution	Default Window having 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.3

(i)Name of item	Password
(ii)Description of purpose	Password of user to identify him as an authenticated user
(iii)Source of input	RTAS
(iv)Valid range & accuracy	20 Characters(max)
(v)Units of measurement	Alphabets ,numbers
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs the	Allows user to log into & access auction system

(viii)Screen format	2 input fields,1 push button,3 links
(ix)Data format	Data displayed as “*****.”
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.4

(i)Name of item	Login
(ii)Description of purpose password	allows user to login if userid & are correct
(iii)Source of input	NA
(iv)Valid range & accuracy	NA
(v)Units of measurement	NA
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs the	Allows user to log into & access the auction system
(viii)Screen format	2 input fields,1 push button,3 links
(ix)Data format	NA
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message the	The “Wrong” message is displayed if userid or password is not correct

No.5

(i)Name of item	Forgot Password
(ii)Description of purpose he	Allows user to get a new password if Forgets his boli password
(iii)Source of input	NA
(iv)Valid range & accuracy	NA
(v)Units of measurement	NA
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs the	Allows user to log into & access the auction system
(viii)Screen format	2 input fields,1 push button,3 links
(ix)Data format	NA
(x)Window format	Default Window having resolution 640 x 480

(xi)Command format	NA
(xii)End Message	The “NEW PASSWORD” message is displayed

No.6

(i)Name of item	OK
(ii)Description of purpose	Acts as enter ie if the user agrees with the given data and wants to proceed
he	presses this button
(iii)Source of input	NA
(iv)Valid range & accuracy	NA
(v)Units of measurement	NA
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs	NA
(viii)Screen format	1 push button in 5 screens
(ix)Data format	NA
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.7

(i)Name of item	Home
(ii)Description of purpose	Allows user to go to the HOMEPAGE
(iii)Source of input	NA
(iv)Valid range & accuracy	NA
(v)Units of measurement	NA
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs	NA
(viii)Screen format	1 link in 10 screens
(ix)Data format	NA
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.8

(i)Name of item	Name
(ii)Description of purpose	Name of the user
(iii)Source of input	NA
(iv)Valid range & accuracy	characters(max 40)
(v)Units of measurement	Alphabets and ‘.’ only
(vi)Timing	Infinite

(vii)Relationship to other inputs/outputs details	Allows user to enter his profile details
(viii)Screen format	6 text fields,3 popup menus,2 push button,2 links
(ix)Data format from	Data displayed as the values input from keyboard
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.9

(i)Name of item	Email ID
(ii)Description of purpose	It the id to where all the emails to the user will be sent
(iii)Source of input	NA
(iv)Valid range & accuracy	characters/numbers/underscore@mailprovider.com
(v)Units of measurement	characters , numbers and underscore
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs details	Allows user enter his profile details
(viii)Screen format	6 text fields,3 popup menus, 2 push button,2 links
(ix)Data format	Data displayed as the values input from keyboard
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.10

(i)Name of item	Confirm ID
(ii)Description of purpose	to confirm the entered ID
(iii)Source of input	NA
(iv)Valid range & accuracy	characters/numbers/underscore@mailprovider.com
(v)Units of measurement	characters , numbers and underscore
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs details	Allows user enter his profile details
(viii)Screen format	6 text fields,3 popup menus, 2 push button,2 links

(ix)Data format	Data displayed as the values input from keyboard
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.11

(i)Name of item	Boli ID
(viii)Screen format	6 text fields,3 popup menus, 2 push button,2 links

Rest Details same as UserID

No.12

(i)Name of item	Boli Password
(viii)Screen format	6 text fields,3 popup menus, 2 push button,2 links

Rest Details same as Password

No.13

(i)Name of item	Date of Birth
(ii)Description of purpose	Date of Birth of user (Security information)
(iii)Source of input	NA
(iv)Valid range & accuracy	Numbers only .1-31 for days depending on the month 1-12 for months and 1950-1990 for years
(v)Units of measurement	dd-mm-yyyy
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs	Allows user to enter his profile details
(viii)Screen format	6 text fields,3 popup menus, 2 push button,2 links
(ix)Data format	Data displayed as the values input from keyboard
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.14

(i)Name of item	Accept terms and submit
-----------------	--------------------------------

(ii)Description of purpose	User presses this button after entering his details & agreeing to the conditions
(iii)Source of input	of bol boli bol
(iv)Valid range & accuracy	NA
(v)Units of measurement	NA
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs	Allows new user to be created
(viii)Screen format	6 text fields,3 popup menus, 2 push button,2 links
(ix)Data format	Data displayed as the values input from keyboard
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	The "User Created" message is displayed

No.15

(i)Name of item	Cancel
(ii)Description of purpose	User presses this button if the wants to go back one page
(iii)Source of input	NA
(iv)Valid range & accuracy	NA
(v)Units of measurement	NA
(vi)Timing	Infinite
(vii)Relationship to other inputs/outputs	NA
(viii)Screen format	1 push button in 8 screens
(ix)Data format	NA
(x)Window format	Default Window having resolution 640 x 480
(xi)Command format	NA
(xii)End Message	NA

No.16

(i)Name of item	Help
(ii)Description of purpose	To provide help to the user on any topic the user wants
(viii)Screen format	1 link in 8 screens
Rest details same as that of the previous item	

No.17

(i)Name of item	Start Selling
-----------------	----------------------

(ii)Description of purpose Allows a user to put up his items for auction
(viii)Screen format 10 links in homepage
Rest details same as that of the previous item

No.18

(i)Name of item **View Cart**
(ii)Description of purpose Contains the items available for bidding
Rest details same as that of the previous item

No.19

(i)Name of item **Buyer View**
(ii)Description of purpose Allows user to see his details as a buyer
(iii)Source of input RTAS
(vii)Relationship to other inputs/outputs Depends on the bids won ,lost and balance of the user
(viii)Screen format 10 links in homepage
Rest details same as that of the previous item

No.20

(i)Name of item **Seller View**
(ii)Description of purpose Allows user to see his details as a seller
(iii)Source of input RTAS
(vii)Relationship to other inputs/outputs Depends on the items sold 7 displayed by the user
(viii)Screen format 10 links
Rest details same as that of the previous item

No.21

(i)Name of item **Update Profile**
(ii)Description of purpose Allows user to make changes to his profile entries
(iii)Source of input RTAS
(xii)End Message The “Updated” message is displayed
Rest details same as that of the item ‘View Cart’

No.22

(i)Name of item **Update Password**
(ii)Description of purpose Allows user to make changes to his password entry
(iii)Source of input RTAS

(xii)End Message The “Updated” message is displayed
Rest details same as that of the item ‘View Cart’

No.23

(i)Name of item **Update email id**
(ii)Description of purpose Allows user to make changes to his email id entry
(iii)Source of input RTAS
(xii)End Message The “Updated” message is displayed
Rest details same as that of the item ‘View Cart’

No.24

(i)Name of item **Logout**
(ii)Description of purpose Allows the user to logout of bol boli bol link in 7 screens
(viii)Screen format
(xii)End Message The “logout” message is displayed
Rest details same as that of the item ‘View Cart’

No.25

(i)Name of item **Favorites**
(ii)Description of purpose Contains the categories of items
(viii)Screen format 1 push button,8 links
Rest details same as that of the item ‘View Cart’

No.26

(i)Name of item **Hot Selling**
(ii)Description of purpose Contains items which are most bided
(viii)Screen format 1 push button,8 links
Rest details same as that of the item ‘View Cart’

No.27

(i)Name of item **Ending Soon**
(ii)Description of purpose Contains items whose bidding is going to end soon
(viii)Screen format 1 push button,8 links
Rest details same as that of the item ‘View Cart’

No.28

(i)Name of item **Second Hand Goods**
(ii)Description of purpose Contains second hand items for bidding
(viii)Screen format 1 push button,8 links
Rest details same as that of the item ‘View Cart’

No.29

(i)Name of item

Paisa Auction

(ii)Description of purpose

Contains items for which bidding can start from any price

(viii)Screen format

1 push button,8 links

Rest details same as that of the item 'View Cart'

No.30

(i)Name of item

1 Paisa Auction

(ii)Description of purpose

Contains items for which bidding starts from 1 paisa

(viii)Screen format

1 push button,8 links

Rest details same as that of the item 'View Cart'

No.31

(i)Name of item

List of Items

(ii)Description of purpose

Contains list of items which a particular seller is auctioning
RTAS

(iii)Source of input

2 links, 3 pop up menus

(viii)Screen format

Rest details same as that of the item 'View Cart'

No.32

(i)Name of item

Payment Preferences

(ii)Description of purpose

Tells how the user wants the payment to be done

(viii)Screen format

2 links, 3 pop up menus

Rest details same as that of the item 'View Cart'

No.33

(i)Name of item

Extra Charges(ii)Description of purpose
extraTells the user if he has to pay any
charges for delivery etc. along

with the

actual payment

(iii)Source of input

RTAS

(viii)Screen format

2 links, 3 pop up menus

Rest details same as that of the item 'View Cart'

No.34

(i)Name of item

(ii)Description of purpose

(iii)Source of input

(iv)Valid range & accuracy

(v)Units of measurement

(viii)Screen format

Rest details same as that of the item 'View Cart'

Current Bid History

Tells the user the details/history of the last bid made

RTAS

can be any number

real numbers(for price)

NA

No.35

(i)Name of item

(ii)Description of purpose

(iii)Source of input

(iv)Valid range & accuracy

(v)Units of measurement

(vi)Timing

(viii)Screen format

Rest details same as that of the item 'View Cart'

Bid Win

Tells the user the bids won by him

RTAS

can be any number

number

Infinite

NA

No.36

(i)Name of item

(ii)Description of purpose

(iii)Source of input

(iv)Valid range & accuracy

(v)Units of measurement

(vi)Timing

(viii)Screen format

Rest details same as that of the item 'View Cart'

Bid Lost

Tells the user the bids LOST by him

RTAS

can be any number

number

Infinite

NA

No.37

(i)Name of item

(ii)Description of purpose

(iii)Source of input

(iv)Valid range & accuracy

(v)Units of measurement

(vi)Timing

(viii)Screen format

Rest details same as that of the item 'View Cart'

Balance

Tells the user his balance

RTAS

can be any number

real number(to tell the amount owned by the user)

Infinite

NA

No.38

(i)Name of item

(ii)Description of purpose

Address

Tells the address of the seller

(iii)Source of input	RTAS
(iv)Valid range & accuracy	Can be anything
(v)Units of measurement	characters, numbers & special
	symbols
(vi)Timing	Infinite
(viii)Screen format	1 push button,1 textarea,,3
	pop up menus,3 text fields, 3 links

Rest details same as that of the item 'View Cart'

No.39

(i)Name of item	Country
(ii)Description of purpose	Tells the Country of the seller
(iii)Source of input	RTAS
(iv)Valid range & accuracy	Has to be a valid , existing
	country
(v)Units of measurement	characters only

Rest details same as that of the item 'Address'

No.40

(i)Name of item	City
(ii)Description of purpose	Tells the city of the seller
(iii)Source of input	RTAS
(iv)Valid range & accuracy	Has to be a valid , existing
	country
(v)Units of measurement	characters only

Rest details same as that of the item 'Address'

No.41

(i)Name of item	Zip Code
(ii)Description of purpose	Tells the zip code of the seller
(iii)Source of input	RTAS
(iv)Valid range & accuracy	Has to be a valid , existing code
	numbers only

Rest details same as that of the item 'Address'

No.42

(i)Name of item	Contact Time
(ii)Description of purpose	Tells the time at which the seller
	can be contacted
(iii)Source of input	RTAS
(iv)Valid range & accuracy	Has to be a valid time
	numbers in time format

Rest details same as that of the item 'Address'

No.43

(i)Name of item	STD Code
(ii)Description of purpose	Tells the std code of the seller
(iii)Source of input	RTAS
(iv)Valid range & accuracy code	Has to be a valid , existing std
(v)Units of measurement	numbers only
Rest details same as that of the item 'Address'	

No.44

(i)Name of item	Telephone Number
(ii)Description of purpose seller	Tells the telephone number of the
(iii)Source of input	RTAS
(iv)Valid range & accuracy	Has to be a valid, existing telephone number
(v)Units of measurement	numbers only
Rest details same as that of the item 'Address'	

No.45

(i)Name of item	Select Item
(ii)Description of purpose	Selects an item for selling
(viii)Screen format	1 push button, 1 textarea, 3 pop up menus, 3 text fields, 3 links
Rest details same as that of the item 'View Cart'	

No.46

(i)Name of item	Condition
(ii)Description of purpose	Tells the condition of the item
(iii)Source of input	RTAS
(viii)Screen format	1 push button, 3 pop up menus, 9 links
Rest details same as that of the item 'View Cart'	

No.47

(i)Name of item	Warranty Options
(ii)Description of purpose item	Tells the warranty offered with the
Rest details same as that of the previous item	

No.48

(i)Name of item	Item
(ii)Description of purpose	It is the commodity to be bided for or displayed for auctioning
(viii)Screen format	1 picture,1 link

Rest details same as that of the previous item

No.49

(i)Name of item

Picture

(ii)Description of purpose

It shows the picture of the item

Rest details same as that of the previous item

No.50

(i)Name of item

Type

(ii)Description of purpose

It describes the type of item

Rest details same as that of the previous item

No.51

(i)Name of item

Current Price

(ii)Description of purpose

It is the current price of the item

(iv)Valid range & accuracy

Can be any number

(v)Units of measurement

real numbers only

Rest details same as that of the previous item

No.52

(i)Name of item

Ends

(ii)Description of purpose

It tells when bidding closes for an item

(iv)Valid range & accuracy

any Date and time format

(v)Units of measurement

numbers only

Rest details same as that of the previous item

No.53

(i)Name of item

Bid Now

(ii)Description of purpose

To make a bid the user presses this button/link

Rest details same as that of the item 'View Cart'

No.54

(i)Name of item

Starting Bid

(ii)Description of purpose
item

Tells the starting bid amount for an

(iv)Valid range & accuracy

any number

(v)Units of measurement

real numbers only

(viii)Screen format

1 push button, 2 pop up menus,
5 links,2 text field

Rest details same as that of the item 'View Cart'

No.55

(i)Name of item	Indicative MP
(ii)Description of purpose	Tells the market price of an item
(iv)Valid range & accuracy	any number
(v)Units of measurement	real numbers only
Rest details same as that of the item 'Starting Bid	

No.56

(i)Name of item	Time left
(ii)Description of purpose	Tells the time left for the bidding to close for an item
(iv)Valid range & accuracy (format)	any valid time (in the time numbers only
(v)Units of measurement	any number
(iv)Valid range & accuracy	real numbers only
(v)Units of measurement	real numbers only
Rest details same as that of the item 'Starting Bid	

No.57

(i)Name of item	Your Maximum Bid
(ii)Description of purpose user	Tells the maximum bid made by the user
(iv)Valid range & accuracy	Can be any number
(v)Units of measurement	real numbers only
(iv)Valid range & accuracy	any number
(v)Units of measurement	real numbers only
Rest details same as that of the item 'Starting Bid	

No.58

(i)Name of item	Quantity
(ii)Description of purpose wants	The quantity of an item the user wants
(iv)Valid range & accuracy	Can be any number
(v)Units of measurement	numbers only
(iv)Valid range & accuracy	any number
(v)Units of measurement	real numbers only
Rest details same as that of the item 'Starting Bid	

No.59

(i)Name of item	Item Description
(ii)Description of purpose	Tells the details of the item
(iii)Source of input	RTAS

(iv)Valid range & accuracy any number
(v)Units of measurement real numbers only
Rest details same as that of the item 'Starting Bid'

No.60

(i)Name of item Seller ID
(ii)Description of purpose Id of the seller of the item
(iii)Source of input RTAS
(iv)Valid range & accuracy any number
(v)Units of measurement real numbers only
Rest details same as that of the item 'Starting Bid'

No.61

(i)Name of item Seller Location
(ii)Description of purpose Address of the seller
(viii)Screen format 1 push button,2 pop up menus,
5 links,2 text field
Rest details same as that of the item 'Address'

No.62

(i)Name of item Member Since(Seller)
(ii)Description of purpose Tells since when the seller has been
a member of bol boli bol
(iii)Source of input RTAS
(iv)Valid range & accuracy Can be any valid date and/or
time format
(v)Units of measurement numbers only
Rest details same as that of the item 'Starting Bid'

No.63

(i)Name of item Place a Bid
(viii)Screen format 1 push button,2 pop up menus,
5 links,2 text field
Rest details same as that of the item 'Bid Now'

No.64

(i)Name of item Model
(ii)Description of purpose tells the model of the item
(iii)Source of input RTAS
(iv)Valid range & accuracy can be any combination of
alphabets
& numbers
(v)Units of measurement alphabets & numbers
(viii)Screen format 3 links,7 text field

Rest details same as that of the item 'View Cart'

No.65

(i)Name of item	Features
(ii)Description of purpose	Tells the features of the item
(iii)Source of input	RTAS

Rest details same as that of the item 'Model'

No.66

(i)Name of item	Memory
(ii)Description of purpose	It is an optional feature that tells memory requirement of the item
(iii)Source of input	RTAS
(iv)Valid range & accuracy	can be any value in range from KB to MB
(v)Units of measurement	numbers only

Rest details same as that of the item 'Model'

No.67

(i)Name of item	List Item
(ii)Description of purpose	the item is displayed for auctioning
(iii)Source of input	RTAS

Rest details same as that of the item 'Model'

No.68

(i)Name of item	Go back to Cart
(ii)Description of purpose	to go back to cart
(viii)Screen format	2 push buttons ,1 link

Rest details same as that of the item 'View Cart'

4 Functional Requirements

4.1 Validity checks & inputs

Validation checks are made at every input to ensure that for an input to be numeric only, numeric data can only be entered and same is ensured for all the inputs

4.2 Exact sequence of operation

When the user types the URL www.BolBoliBol.com in his address bar of internet explorer the **Login** page opens. Now if the user does not exist then he

selects the **Join Now** link and enters his details, accept the terms and on submitting gets the **User Created** page. Now a mail is sent to his email id and if he clicks the link in that mail, his account is activated and he becomes a **registered user**.

If the user forgets his password then he can select the **Forgot Password** link from login page and can get a new password

If the user is not registered or invalid userid or password is entered in the login page then the **Wrong** page is displayed else the **Home page** is displayed

Now if the user selects **Seller View** he can see his details as a seller and similarly he can see his buyer details in **Buyer View**. If the user wants to sell he selects the **Start Selling** link enters his details, selects an item, gives the item details

and on selecting List Item gets the **Item Listed** page. And if he selects **View Cart** then he can select an item from the given categories and place a bid following which he gets the **Bid Placed** page.

4.3 Error handling & recovery

The system has been aimed to provide to be able to rollback and recover effectively under any crisis or failure. And if required there should be minimum number of restarts The system is designed to respond to certain situations in such a manner that these have minimal effect on the system's performance and efficiency. For instance, the system would either not respond to invalid inputs or it would display messages to guide the user to use valid values.

5 Dynamic Requirements

5.1 Static requirements

No of terminals to be supported: Terminals attached to the server must not exceed 20.

No of simultaneous users to be supported: No Bounds

Type of information to be handled: The information that will be fed or collected by the server via communication or user interaction like date and time of bidding, user details etc. Only part of the information can be retrieved by the user like his profile, buyer view and seller view.

5.2 Dynamic requirements

No of transactions

No bounds

Amount of data to be processed

No Bounds

Type of data to be processed

The data entered through user profile
And the one obtained from the bidding

bided & made and items displayed,
bought

5.3 Human interaction with the system

Query Handling

The only type of query the user can do is he can check his buyer view and seller view Information fed into the system by the user. The user enters his profile into the system and his details as seller along with the details of the item he wants to sell.

6 Logical Database Requirement

Database Name	RTAS
Table Name	User
	Buyer
	Seller
	Item
	Bidresult

Schemas of Tables

User(Name,EmailID,BoliID>Password,DOB)

Buyer(BoliID,Bid win,Bid Lost, Balance,CBHist)

Seller(BoliID,Address,Country,ZipCode,CTime,STDCode,Tel,ItemMod

el)

Item(Model,Type,CPrice,PayPref,Extra,Ends,SBid,YBiD,Condition,Quantity,
SellerBoliID,Features)

Bidresult(Model,BoliIDSeller,BoliIDBuyer,HBid)

7 Design Constraints and Standard Compliance

Report Format: The various reports like BID RESULTS report, ITEMS AUCTIONED report would be generated in the specified formats as per requirements.

Data Naming: NA

Audit tracing:NA

8 Software System Attributes

8.1 Reliability at delivery time

The system is designed taking care of all requirements and situations so as to work correctly always .The system must fulfill all the specified constraints to obtain maximum results

8.2 Availability

The various mechanisms of checkpoints, recovery and restarts have been incorporated into the system to guarantee a defined level of availability for the entire system.

8.3 Security

Cryptography Technique used

DSA

No user can access the information other than his own profile, buyer view, seller view .Although, the results of all bids made by him will be mailed to him but they will contain only the data which will let him know if he had won or lost .In case , of losing he will come to know about the winning and runner ups bids whereas in case of winning he will have to pay for his bid which will be delivered to him accordingly.

8.4 Maintainability

Modularity will be achieved by using functions for doing specific jobs like creating users, buyer view ,seller view, user profile, cart etc.and various other independent jobs.

8.5 Portability:

The entire code is host independent and can be run on any host machine which is authorized to access the system and has the required implementation tools and the operating system installed on it.

17.4 Flow of Events

- 1 The system is accessible to registered users only. So, any user who wants to access the system enters his/her user ID & password.
- 2 If the user is not registered then he can get registered if he joins now.
- 3 If the user is registered then he has the choice of
 - Selling
 - Buying

- Editing details
 - Viewing his profile
- 4 If the user wants to start selling then he must do so in the given fashion
- A) He must enter his seller details
 - B) He must select an item & enter the details of the item he wants to sell
 - C) He must list the item on the site.
- 5 If the user wants to buy then he must follow the following procedure
- A) He must select the item he wants to buy by bidding
 - B) He must place a bid
- 6 If the user wants to view his profile he can do so by clicking on the “Seller Profile” and “Buyer Profile” links.
- 7 If the user wants to edit details then he can do so by clicking on the links “Update Profile”, “Update Password” & “Update email id”
- 8 All the activities of the user must be maintained in the user log for security, privacy non repudiation, and authentication & integrity purposes.
- 9 Also, the transactions details must be maintained in the database .So, after each
and every activity / transaction the database is updated.
- 10 The user can exit from the system by logging out

17.5 USE CASE Diagram

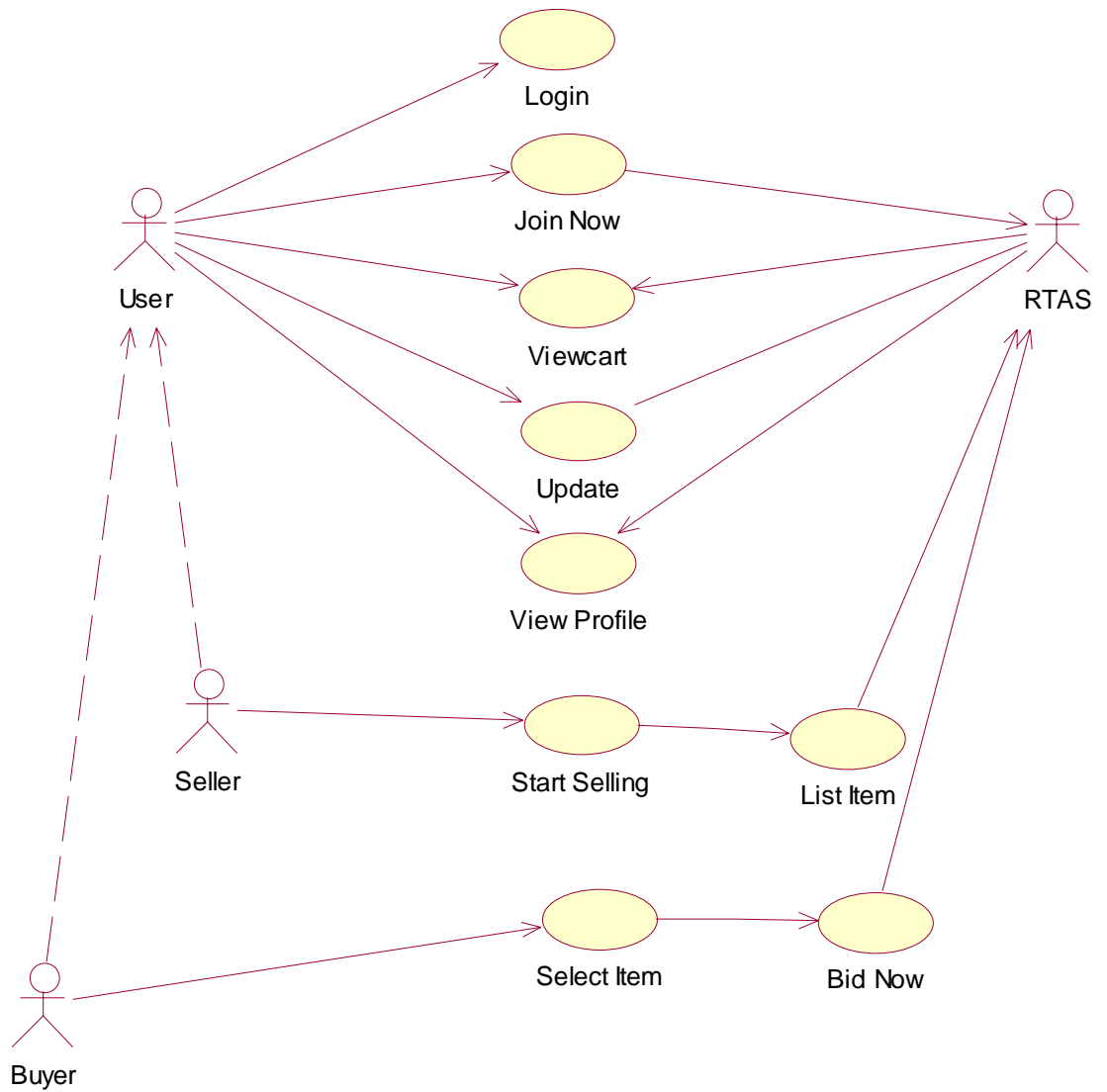


Figure 55: Use Case Diagram

17.6 Class Diagram

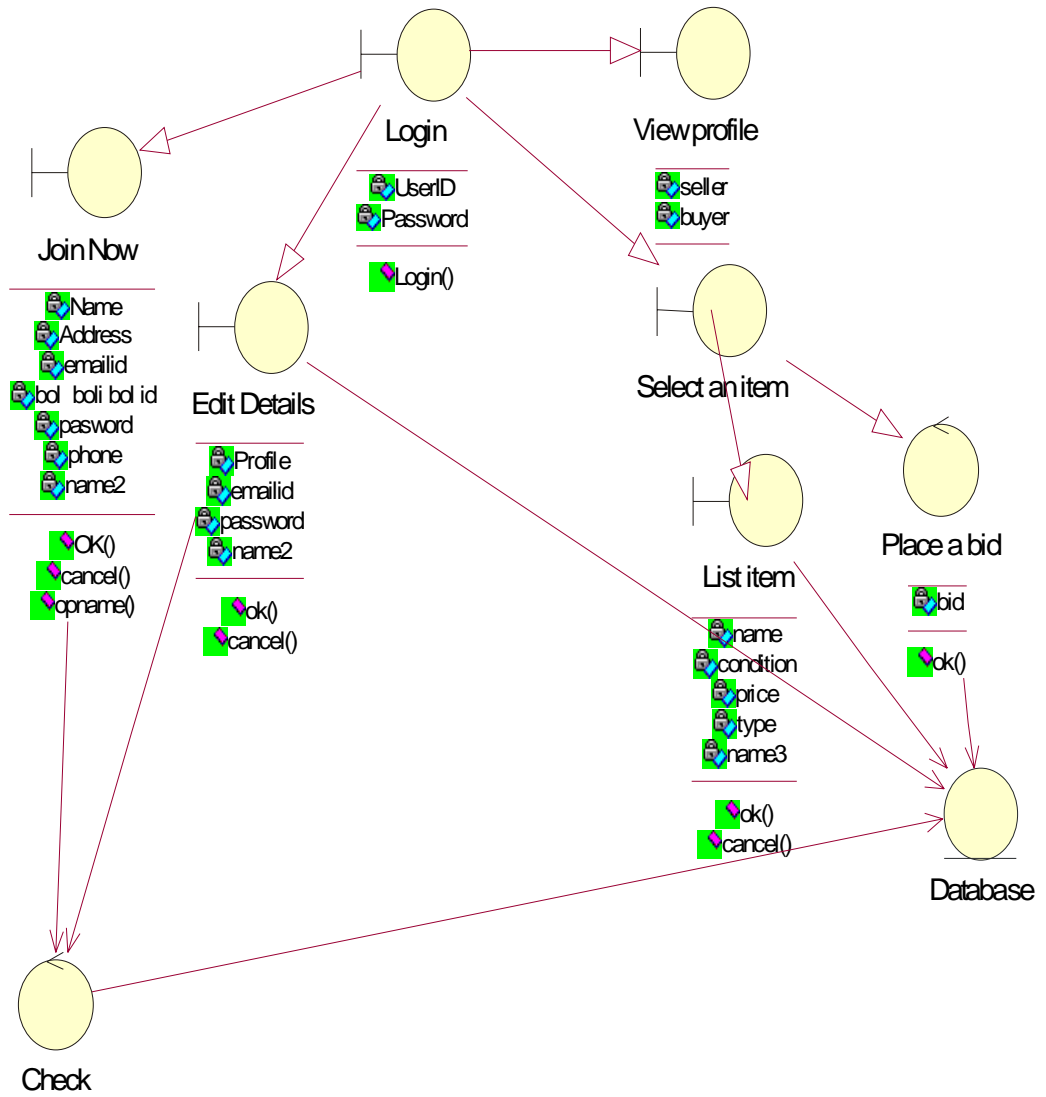


Figure 56: Class Diagram

17.7 Sequence Diagram

17.7.1 Placing a BID:

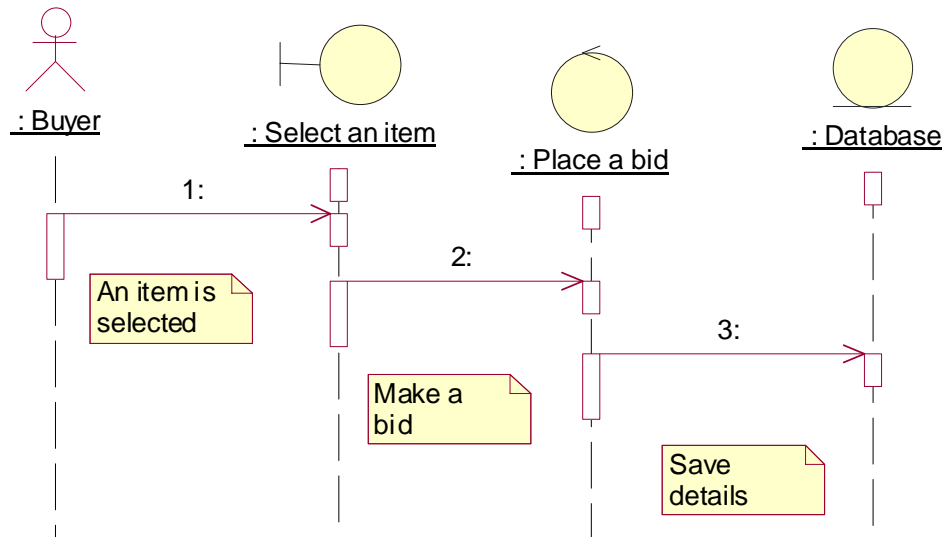


Figure 57: Sequence Diagram for Placing a bid

17.7.2 Join:

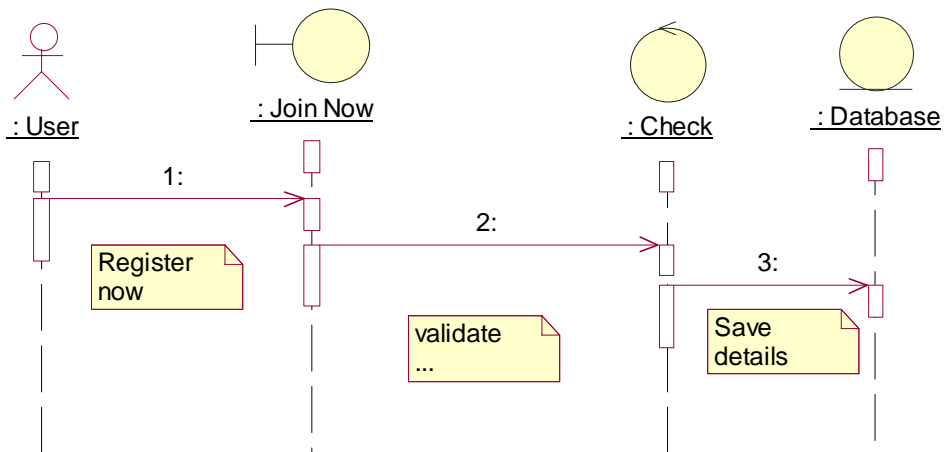


Figure 58: Sequence Diagram for Join

17.7.3 Listing an Item:

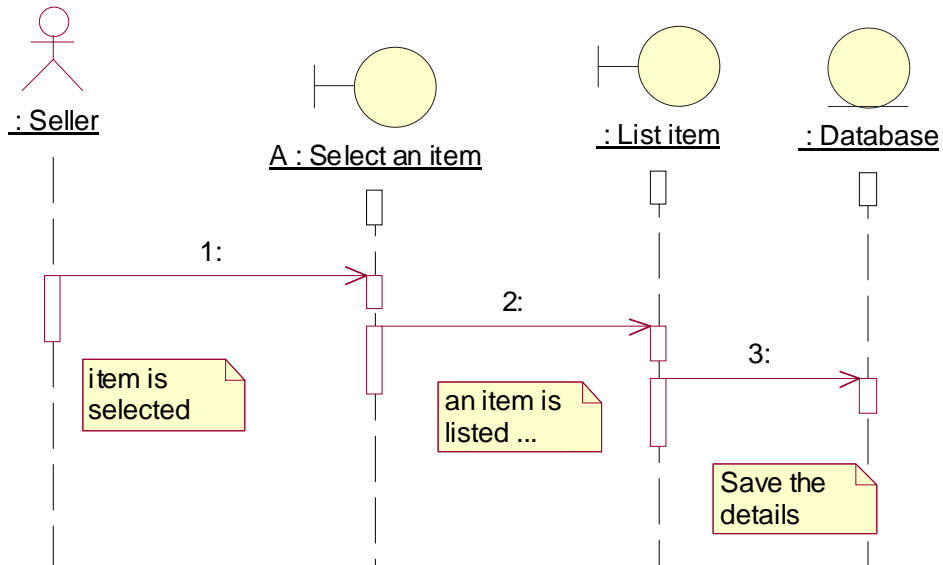


Figure 59: Sequence Diagram for Listing an item

17.7.4 Login by User:

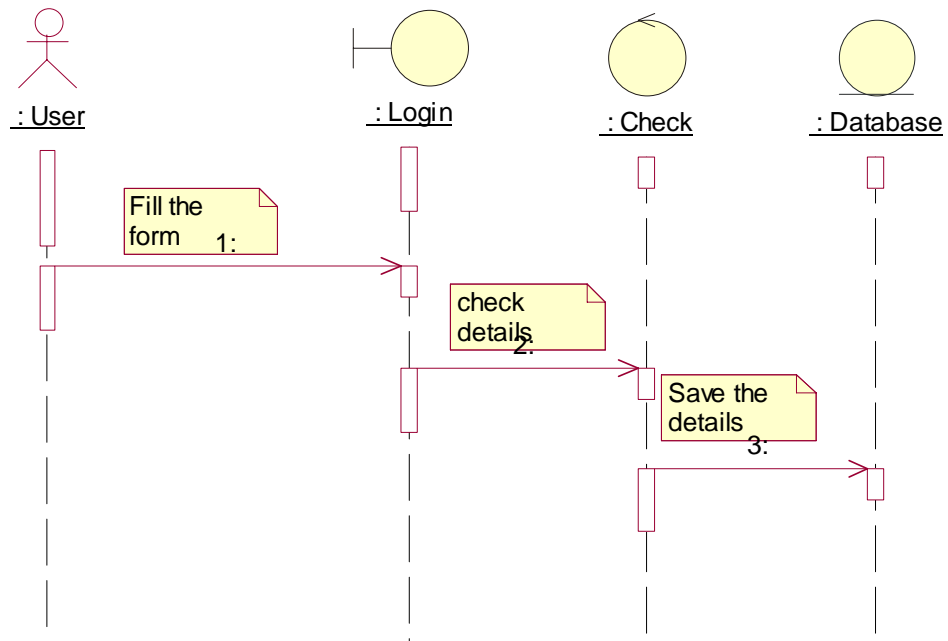


Figure 60: Sequence Diagram for Login

17.7.5 View Profile:

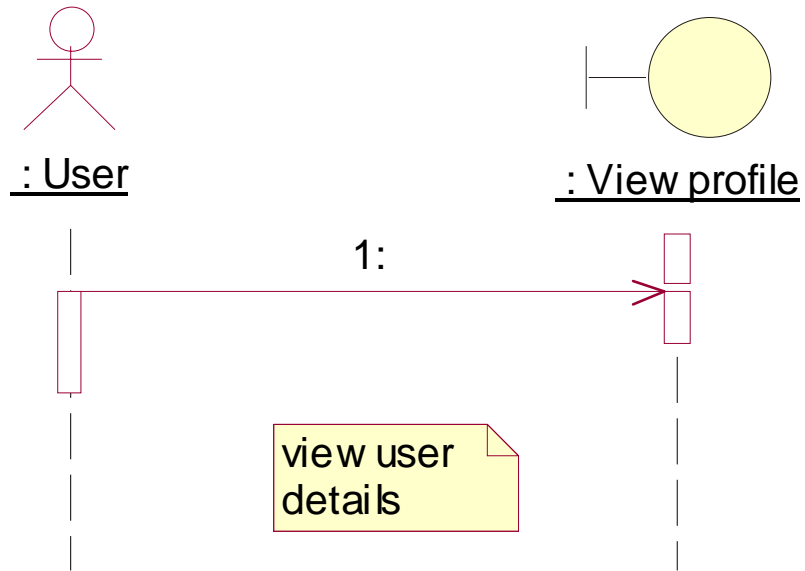


Figure 61: Sequence Diagram for Viewing Profile

17.8 Collaboration Diagram

17.8.1 Placing Bid:

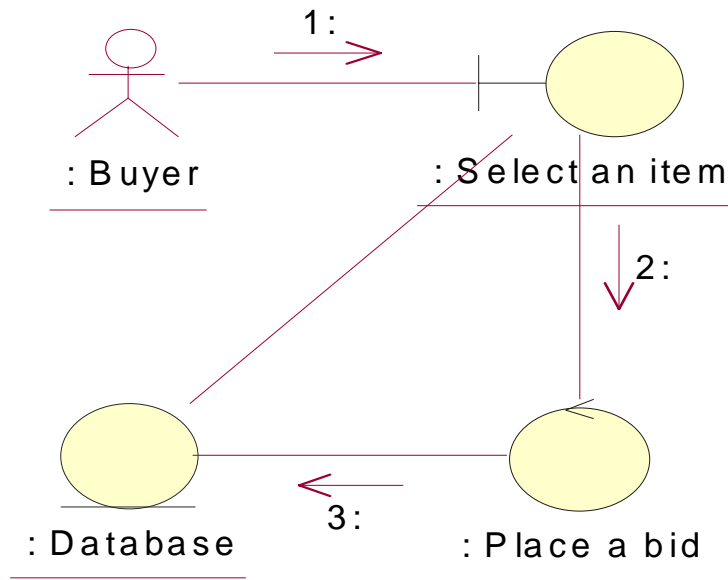


Figure 62: Collaboration Diagram for Placing bid

17.8.2 Join:

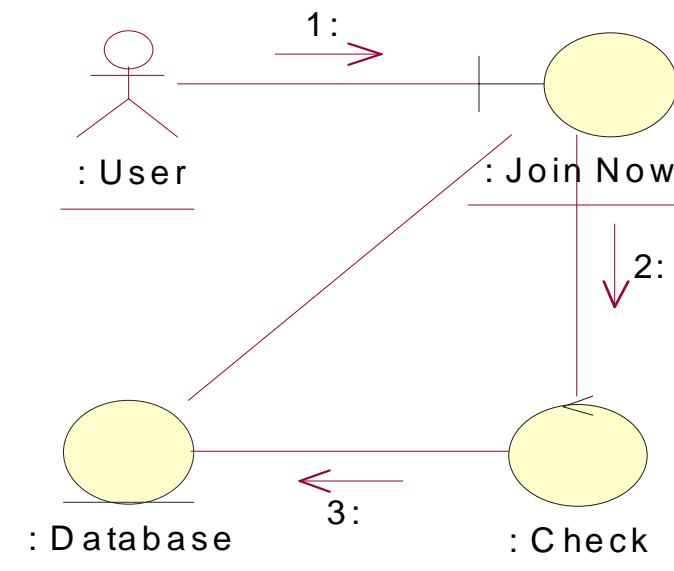


Figure 63: Collaboration Diagram for Join

17.8.3 Listing Item:

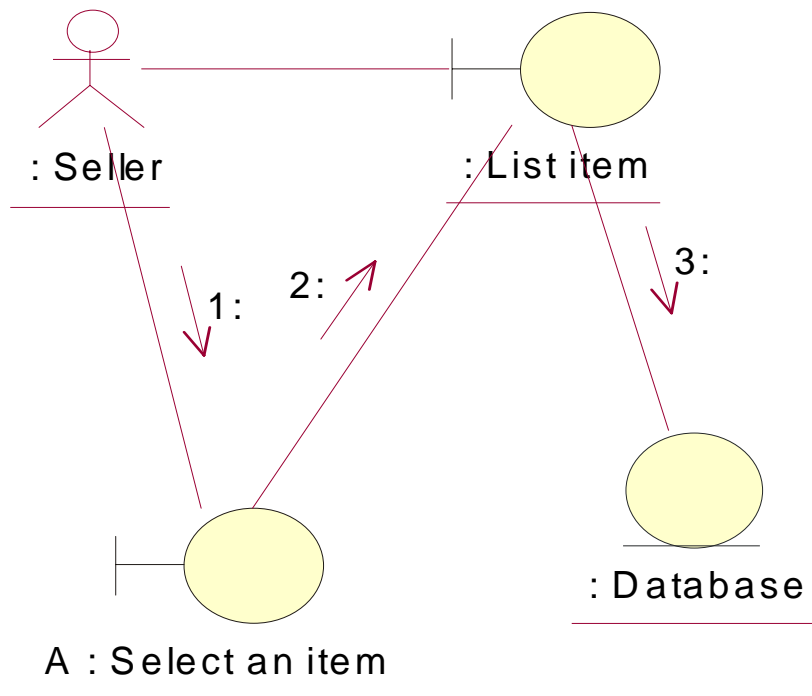


Figure 64: Collaboration Diagram for Listing an item

17.8.4 View Profile:



Figure 65: Collaboration Diagram for Viewing Profile

17.8.5 Login by USER:

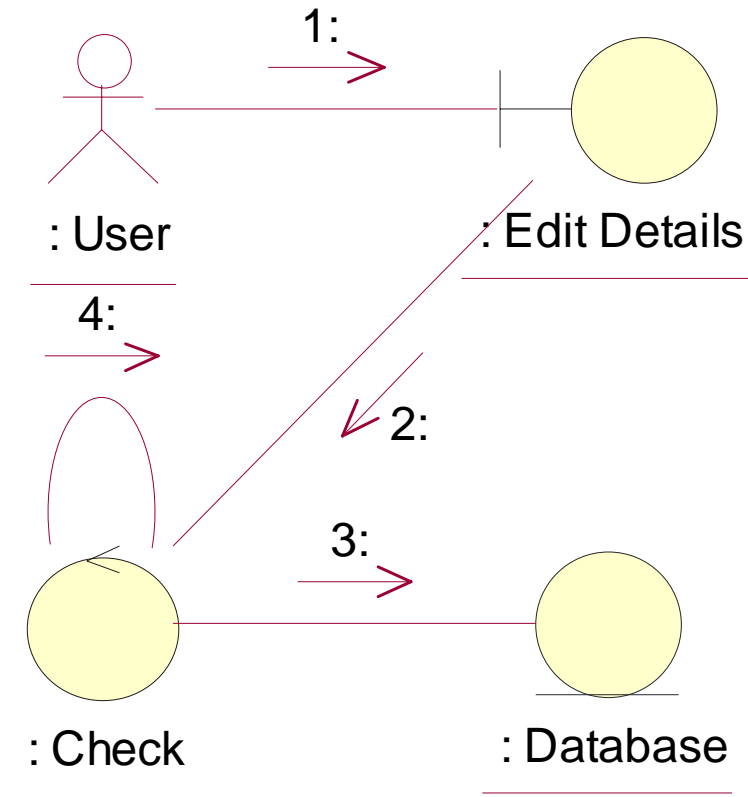


Figure 66: Collaboration Diagram for Login

17.9 Activity Diagram

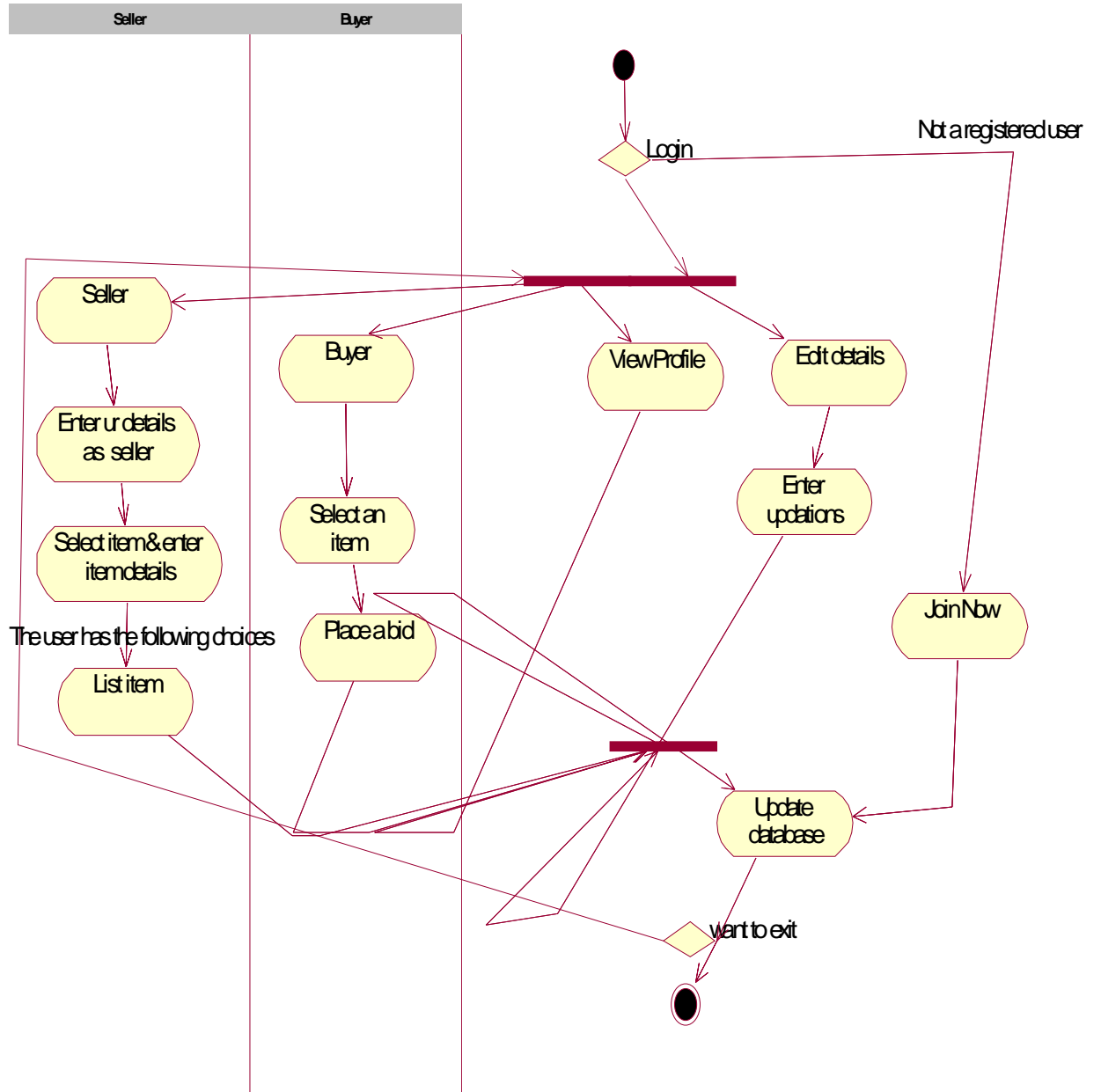


Figure 67: Activity Diagram for Auction System

17.10 State Transition Diagram

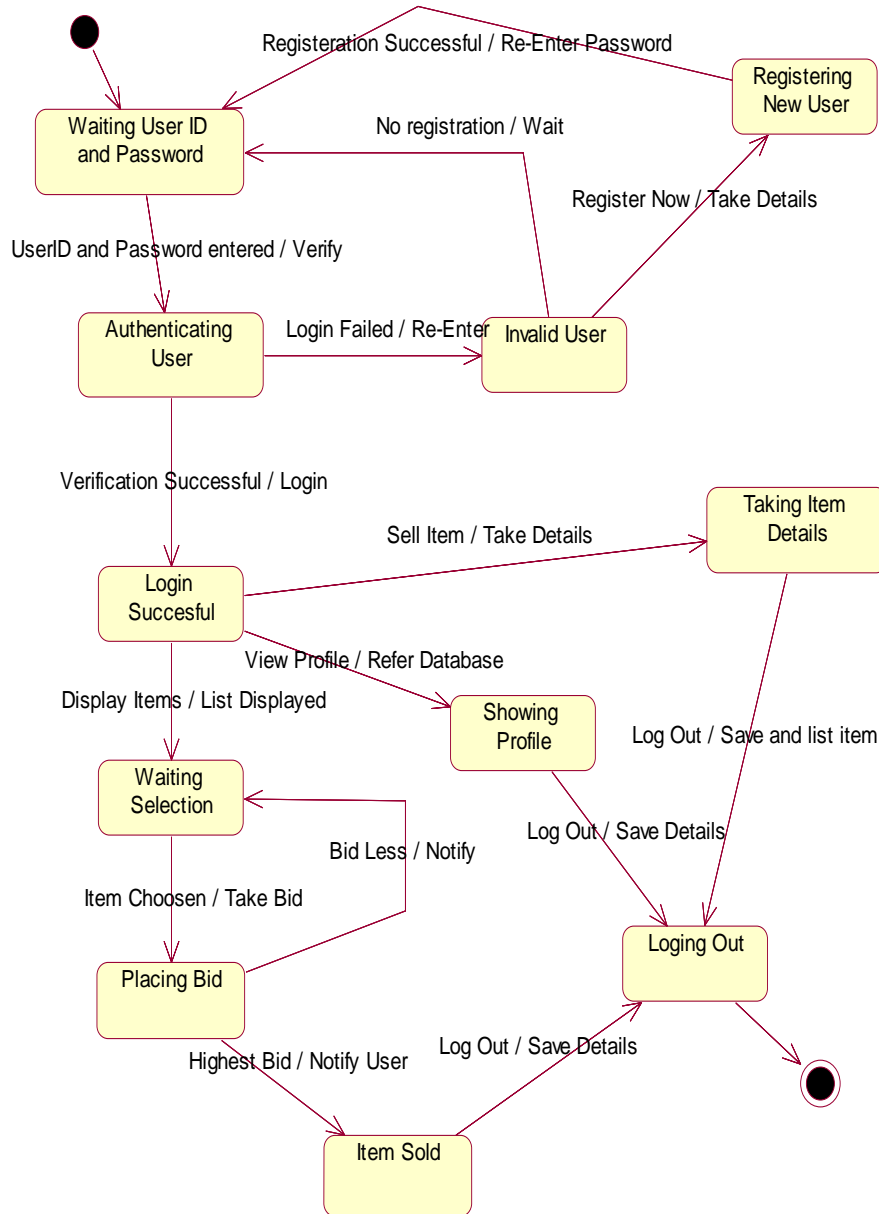


Figure 68: State Transition Diagram for Auction System

17.11 Deployment Diagram

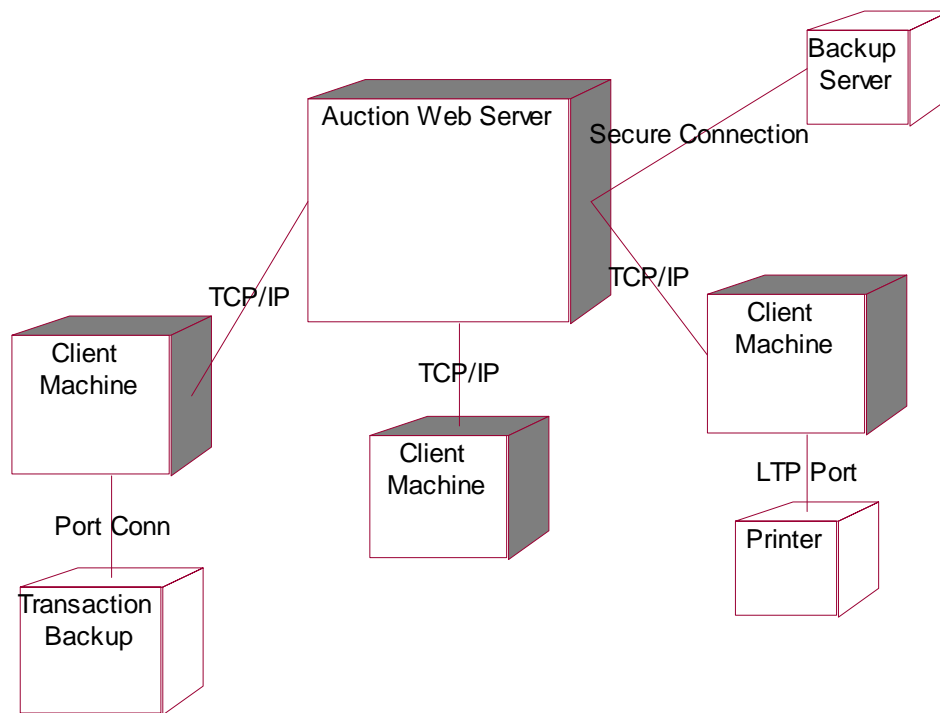


Figure 69: Deployment Diagram for Auction System

CONCLUSION

The Study of Rational Rose Software helped us to gain knowledge of Software Engineering Process and software development steps. It is a tool which makes the probability of failure in a project very less. Visual Modeling and UML Technology lies at the base of Rational Rose. The use of these technologies and converting their strengths in form of nine diagram windows makes the development procedure relatively such simpler. By using Rational Rose, it is easy to understand problems, communicate with everyone involved with the project (customers, domain experts, analysts, designers, etc.), model complex systems; prepare documentation, and design programs and databases. Keeping track of each subunit and giving a continuous workflow in the development and logic makes the development easier. The use of RUP (Rational Unified Process), which is a control for an iterative and incremental life cycle, assures that risk of failure is gradually removed from SDLC (Software Development Life cycle). It assures Sustained delivery of high-quality software and cohesive teamwork with a common understanding of development tasks. Thus software coding after modeling via Rational Rose is much easier and Software thus produced are more reliable, better manageable, having less cost, having distributed cost, having better quality and likely to be successful on real time machines.

REFERENCES

1. Visual Modeling using Rational Rose 2002 and UML
(TMH)
2. Object Oriented Analysis and Design
(Manipal Academy of Higher Education)
3. Software Engineering
(Sangeeta Sabharwal)
4. An Integrated Approach to Software Engineering
(Pankaj Jalote)
5. Using Rose
(Online Rational Manuals)
6. Rose Tutorial 2002
(Online Rational Manuals)
7. Rational Suite Introduction
(Online Rational Manuals)
8. Rational Rose Manuals
8. www.rational.com
9. www.newinstruction.com
10. www.rational.net
11. www.awproffessional.com etc