EVALUATION OF UML TOOLS USING AN END-TO-END APPLICATION

Shibi Mary Thomas

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 2003

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

____

(Signature of Candidate)

_____________ day of ___________________ (year) _________

ABSTRACT

Any software project goes through the different stages of a Software Development Life Cycle (SDLC). Like any other commercial product, software has a design stage but this stage is unique and critical to software due to its soft nature. A system that is given careful thought at the design phase results in a correct and complete system and adheres to software design principle. The “Unified Modelling Language” (UML) is a standard modelling language for object-oriented systems. Many tools are currently available to support the design and implementation of software. Generating skeletal code from a design brings down the implementation time considerably.

This research report presents a list of criteria against which one can compare different UML tools, and puts forward a rating system where decisions can be made on them. It presents a comparison between four UML tools: ArgoUML, Rational Rose, Together Control Centre, and MasterCraft. An end-to-end application was developed on each of these tools as part of the evaluation process. During the design phase a detailed design was done using the ICONIX process. The different features of an ideal UML tool is analysed and used to evaluate the four selected tools. Of the four tools, Rational Rose, Together Control Centre, MasterCraft are off-the-shelf modelling softwares whereas ArgoUML is an open source modelling software. From the evaluation it is observed that Together Control Centre attains a high score with Rational Rose following just behind. MasterCraft comes third. Argo UML has the least score but it has the advantage of being an open source software.

ACKNOWLEDGEMENTS

This dissertation and the resulting prototype described herein, was completed within the Information Engineering Research Programme (IERP) at the University of the Witwatersrand.

First and foremost I would like to thank my God who was with me through all the moments of this research work.

I would like to acknowledge Prof B. Dwolatzky for his supervision of this study and for his unwavering and consistent support throughout the duration of this project. I would also like to thank him for making available the resources to proceed with my work.

I would like to thank TATA Consultancy Services (Pune, India) for the support and training arrangements, they provided for their tool, MasterCraft, to abet my work.

I would also like to express my gratitude to the members of the IERP group for the small technical support and advice that has added value and helped me proceed in my research work.

Last but not least, I would like to acknowledge the support and unwavering encouragement of my family throughout my studies with boundless enthusiasm for all my chosen endeavours.

TABLE OF CONTENTS

FOREWORD

This MSc Research project report is different from the conventional format in that it comprises of a paper and a number of appendices. It is therefore considered helpful to provide the reader the guidance regarding the order in which the various documents should be reviewed. There are various UML concepts used in this project report. It is assumed that the reader is familiar with basics of UML and its concepts.

The Project Overview gives the user an overall description of this research project. This gives a background of how this project came about and why it is important. It also gives a brief description of the work done in this project and the outcome of this research work.

For the essence of the project, the reader is directed to the paper entitled

"*Evaluation of UML tools using an End-to-End Application*".

The substance of the project will be found in this paper, and the appendices should be regarded as additional sources of information in understanding the issues at hand.

Discussion and Conclusion contains important information regarding the conclusions that were reached. This is followed by a list of all the references used for this work.

The appendices to the research project are discussed below.

Appendix I discusses how the application was taken through the different stages of the software development cycle. It focuses primarily on the design phase. The design phase of the application is elaborated further by going through the steps of the ICONIX process.

Appendix II talks about the evaluation methodology used for the evaluation of tools. This section discusses the criteria defining each feature and sub features that an ideal UML tool should have. The approach used towards rating each tool, is also discussed here. The evaluation process breaks down each feature and quantifies the evaluation using a rating method.

Appendix III discusses all the UML tools used in this research work. This section brings forth some of the additional feature that each tool has and hence making them unique in their own way. It also talks about the features that should be part of the tool.

Appendix IV discusses the general interfaces of the tools. This section talks about each tool discussing the usability of the tool. The functions provided for navigation, the different panes etc are also discussed in this section. It also talks about modelling class diagrams and sequence diagrams in the different tools.

Appendix V concentrates mainly on the function round trip engineering. This section compares the code generation and reverse engineering in the various tools. Appendix VI gives the final evaluation table.

PROJECT OVERVIEW

Throughout the Software Development Life Cycle (SDLC), seamlessness is both a highly desirable and much sought after feature. Like any other commercial product, software has a design stage but this stage is unique and critical to software due to its soft nature. A system that is given careful thought at the design phase results in a correct and complete system and adheres to software design principle. Achieving an effortless transition from the requirements and analysis phase to the design phase is a complicated task. The ability to harness the power of design offered by the Unified Modelling Language (UML) into a tool, which provides a high level of usability and functionality, would be a considerable advantage. Some of the functionalities incorporated in the tool, empowers the implementation and testing of the design developed. Code generation in an accurate manner, is significant since it reduces the time of the implementation phase and hence the cost.

The Unified Modelling Language, an Object Management Group (OMG) standard since 1997, is a visualizing language for the modelling and development of software systems. The UML offers a standard way to write a system's blueprints, including conceptual aspects such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.

Since the official release of UML in late 1997, the number of commercial UML modelling Computer Aided Software Engineering (CASE) tools has increased dramatically in the market. This provides us with many choices, yet requires us to select the right UML modelling tool that best meets our business and software application development requirements and achieves the best Return on Investment (ROI). The UML modelling CASE tools enable us to apply a formal object oriented analysis and design methodology and abstract away from the entanglement of source code, to a level where architecture and design become more apparent and easier to understand and modify. As the systems being built today become more and more complex, UML modelling CASE tools offer many benefits for everyone involved in a project, e.g., project manager, analysts, designers, architects, developers and so on. UML tools play a significant role in developing software systems better, faster and in a cost effective way.

The research presented in this report defines a list of criteria against which one can compare different UML tools, and puts forward a rating system where decisions can be made on them. The usability aspects and functional aspects of ideal UML tool were studied. Four different tools (ArgoUML, Rational Rose, Together Control Centre, and MasterCraft) were chosen to evaluate the criteria. An example application with sufficient complexity was taken to walk through the different stages of a SDLC using each of the above mentioned tools.

UML on its own is a mere modelling language but it is necessary to put a process in place while doing the design. In this report the process used while designing the system is the ICONIX process [10, 11, 12, 42]. The rating system used in this research gives an analyst, who carries out a comparison between various tools, the flexibility to prioritize his features according to the requirements. Hence the matrix [appendix VI] presented in this project report could be used as a comparison tool to compare any set of UML tool.

This evaluation tool or the matrix emphasizes, particularly on the technical features in a UML tool. These technical features are mainly functional and usability issues. Non-technical aspects such as cost, training, local support, availability of resource and developmental environment are not part of the comparison matrix. Choosing a tool is critical and requires a lifetime investment yet design is one of the aspects that cannot be lightly tinkered with. This work aids in rating a tool and hence selecting an appropriate tool.

PAPER I

EVALUATION OF UML TOOLS USING AN END-TO-END APPLICATION

EVALUATION OF UML TOOLS USING AN END-TO-END APPLICATION

Shibi Mary Thomas

Information Engineering Research Group, Information and Electrical Engineering, University of Witwatersrand, Johannesburg, South Africa.

Abstract

Throughout the Software Development Life Cycle (SDLC), seamlessness is both a highly desirable and much sought after feature. A system, which is given careful thought during its design phase, caters for a correct and complete system and covers the basics of the software design principle. The Unified Modelling Language (UML) offers a standard way to write a system's blueprints. The UML modelling CASE tools enable us to apply the formal object oriented analysis and design methodology such that architecture and design become more apparent and easier to understand and modify. There are different tools with diverse degree of functionalities available for modelling software. This paper tries to draw up ideal evaluation criteria for a UML tool and using these criteria, evaluates four UML tools. An application is developed, taking it through the different stages of SDLC, which includes a detailed design and implementation, in an object oriented language. The design and implementation phase of the application are used to evaluate the different features of the tools chosen. The matrix presented in this paper could be used as a comparison tool to compare any set of UML tool.

Keywords
Software Engineering, Object Oriented, UML Tools, Evaluation, Methodology, Rating.

Computing Review Categories
Software Engineering Design, Evaluation of Tools

1. Introduction

Since the Stone Age, tools have been an integral part of man enabling him to simplify and improvise his work. Like other engineering fields software engineering has its own set of tools directed towards specific developmental requirements and areas. Yet unlike other tools, software tools need to adapt to the soft nature (i.e. the continuously changing requirements) of the environment where they will be used.

Since the official release of UML in late

1997, the number of commercial UML modelling CASE tools has increased dramatically in the market [1]. There are many UML tools available in the market where the tool vendor tries to prove their tool better than the others. Hence before a user invests in a UML tool it is essential to consider the business requirements as well as the software application development requirements. The research described in this paper evaluates four different UML tools: ArgoUML v 0.14, Master Craft for Java v 6.0, Rational Rose 2001 and Together Control Center v 6.1. A technical evaluation of the tools was carried out based on an end-to-end application. The evaluation is rated based on the author's experience with each of these tools and a conclusion is drawn from this rating.

2. Software Engineering Principles and Design

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is the application of engineering to software. [2] When constructing software, the problem to be solved is analysed and the requirements are defined in a precise manner.

There are six distinguished phases in the Software Development Life cycle (SDLC):

- Preliminary Investigation Phase (Feasibility Study)
- Analysis Phase
- Design Phase
- Implementation Phase
- Testing Phase
- Maintenance Phase [2] [3].

Any software system, which is given careful thought during its design phase, caters for the factors that lead to a correct and complete system. A correct and a complete software system is a system which has maximum cohesion, minimum coupling, maintainability, reusability, understandability, and adaptability. The characteristics of a system with a good design are:

1. Change in one part of the system doesn't always require a change in another part of the system.
2. Every piece of logic has one and one home. There is no duplication of logic.
3. System can be extended with changes in only one place.
4. Simplicity. [4]

Software is not a jumble of code that can be written and put together by any programmer. Many do not realize the enormity of skills and knowledge required to build a complete and correct software system.

3. Why Object Oriented Design and Programming?

To put the ever-growing power of computers to good use we need software of much greater complexity, yet the software needs to be more reliable. High Quality is essential in software development while poor quality is a waste of time and money. To improve quality, the design has to be given careful thought. There are different types of design: Structured design, data-driven design, object-oriented design, object-based design etc. Object oriented techniques is currently one of the better techniques used to simplify the design of complex systems.

Using the object oriented technology the domain of a system can be visualized as a collection of objects existing in one of the specified states. The operations that change the state are simple. Objects are built out of other objects.

Design can be thought of in two phases. The first, called high-level design, deals with the decomposition of the system into large, complex objects. The second phase is called low-level design. In this phase, attributes and methods are specified at the level of individual objects. This is also where a project can realize most of the reuse of object-oriented products, since it is possible to guide the design so that lower-level objects correspond exactly to those in existing object libraries or to develop objects with reuse potential.

*"High thoughts must have high language"*Aristophanes. [9] Hence to put our complex systems into software we need a powerful language. The implementation in this research is done in Java. Java is a fully object-oriented language with strong support for proper software engineering techniques [9]. Java is a powerful language with various class libraries. It is used to develop internet-based and intranet-based applications and software for devices that communicate over a network.

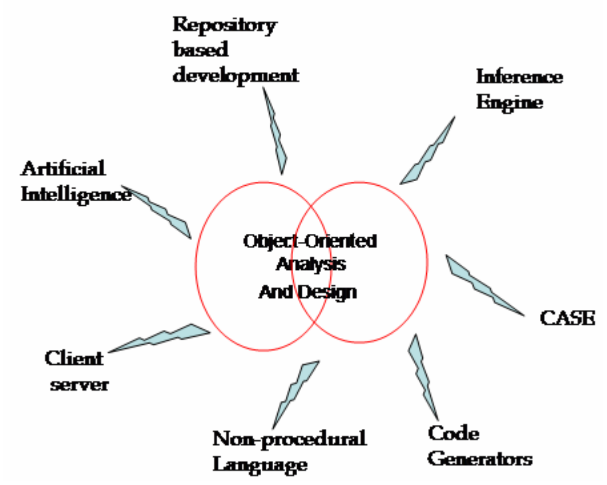


Figure 3.1 Software Technologies

Object Oriented System alone cannot provide the magnitude of change needed. They must be combined with other software technologies. Some of them include repository-based development, code generators, repository based methodologies etc. as shown in Fig. 3.1. [22]

4. Unified Modelling Language

The Unified Modeling Language (UML), an Object Management Group (OMG) standard since1997, is a visualizing language for the modeling and development of software systems [5, 6]. The UML is a modeling language, not a software development process, and it intends to support different object oriented approaches to software production. UML is also a standardized notation for object-oriented analysis and design. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.

UML defines twelve types of diagrams, divided into three categories. The three categories are:

1. Diagram used to represent static application structure or Structural diagrams.
2. Diagrams used to capture the different aspects of dynamic behavior also called Behavior diagrams.
3. Diagrams used to model and manage the application modules. This is also called Model Management diagrams.

Four diagram types represent structural diagram; five represent behavior diagrams; and three represent model management diagrams. *Structural Diagrams* include the Class Diagram, Object Diagram, Component Diagram, and Deployment diagram. *Behavior Diagrams* include the Use-case Diagram (used by some methodologies during requirements gathering); Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram. *Model* Management Diagrams include Packages, Subsystems, and Models. [5, 6, 7, 8]

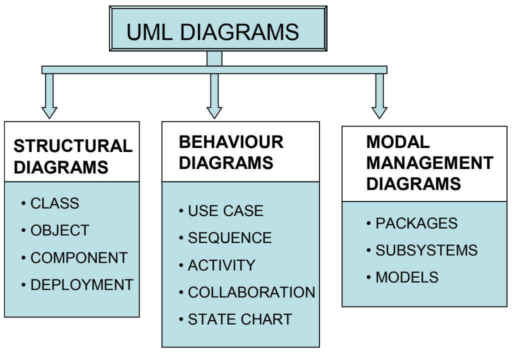


Figure 4.1 Types of UML Diagram

5. ICONIX Process

The visual language UML has to be combined with a process while designing. The process that will be used in this research will be ICONIX. This methodology was created by Doug Rosenberg. ICONIX uses robustness analysis as a bridge between use-cases (or Domain Model) and the code [10]. This methodology assumes an Object Oriented decomposition of the domain and it is use-case driven. The diagram [Fig 5.1] below explains the ICONIX process. It consists of two parts: dynamic and static. The main stages in this are domain modeling, use-case modeling, robustness analysis, interaction modeling and finally class diagram. [11, 12]

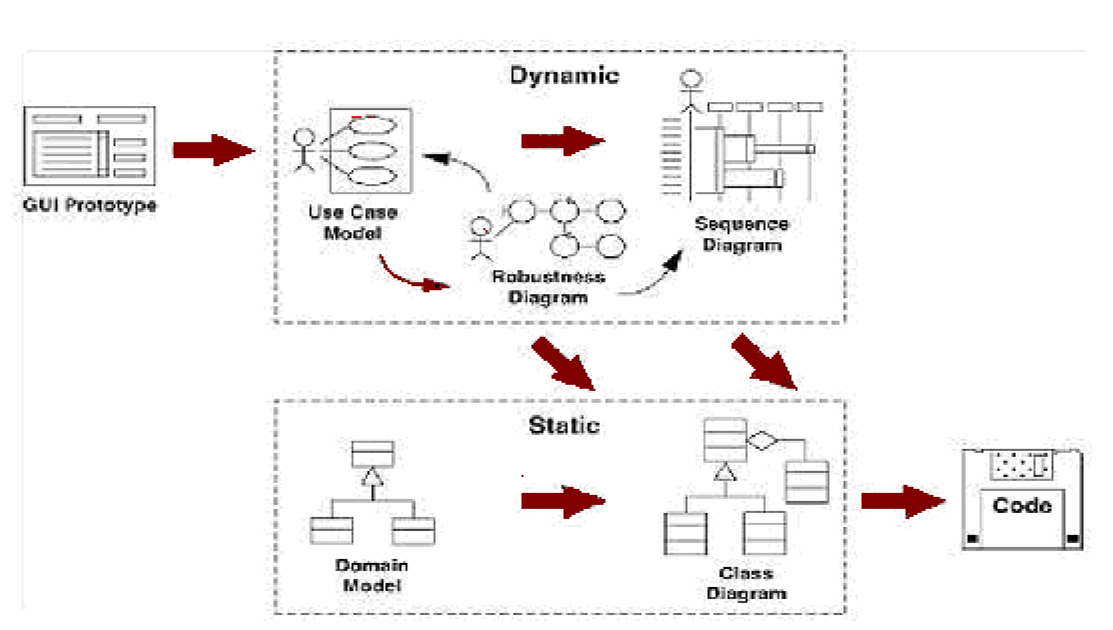


Figure 5.1 ICONIX Processes [12]

6. UML Tools

UML tools help designers to model their design. Most of these tools support: drawing and exporting UML diagrams, eliminating errors, model- and document- linking, report generation, configuration management, code generation and reverse engineering.

The number of UML tools in the market has increased incredibly, since UML was made a standard in 1997. This provides us with a number of choices but at the same time requires us to do more investigations, to select the right UML modeling tool, according to our business needs. OMG suggests some of the features that may be essential in a UML tool like Repository support, Roundtrip engineering, UML support, Pick lists etc [9]. Many products have done a comparison with their competitors to prove that their products are better, hence pointing out some of the features essential in a tool [1, 10, 11, 12].

UML tools can be classified as follows:

- Basic diagram-drawing tools
  E.g. Visio (basic version).
- Main-stream OO CASE tools
  E.g. Together, ArgoUML.
- Specialist real-time/embedded tools.
  E.g. Rhapsody, Telelogic. [13]

7. Evaluation Criteria

Evaluation criteria are selected in order to assess the functionality and usability of the tool as a whole, not merely from a superficial, interface evaluation perspective [14]. It is essential to evaluate everything from the compliance of a tool's notation to the UML standard, to the ease with which it supports navigation between diagrams and ease of use. Evaluation criteria were drawn up to highlight the basic requirements and the future evolving requirements, of an ideal UML tool. Each of these features tries to focuses on the ideal technical facilities provided by the tool rather than the ostensibly appealing features promoted by the vendor.

The evaluation criteria are divided into two sections: Ideal features and other features. The ideal features discuss all features that an ideal tool should have. These features are used as part of the evaluation table (See table 9.1) to evaluate the tools. Other features talk about commonly used features in a tool. Hence it will be good to include them in the tool.

7.1. Ideal Features

7.1.1 Repository

For a large project, a repository is necessary for the sharing of component designs between developers. Two or more developers can share components of a model or even collaborate on the development of a single component by defining ownership and sharing rights at the appropriate level. With modeling tools that support a repository, changes to any component should be automatically propagated to any design which imports the component.

7.1.2 Customisation

It can be useful for a developer to be able to configure the tool to conform to some specific standards, like company requirements, personal preferences etc. This could include the options to view different panes, tools etc.

7.1.3 HTML Documentation

The UML modelling tool should provide seamless generation of HTML documentation of the model. HTML documentation provides a static view of the object model that any developer using the model can refer to quickly in a browser, without having to launch the modelling tool itself. This reduces the number of required licenses for the modelling tool for those who require read-only access to the model information. The HTML documentation should include a bitmap picture of each of the diagrams in the model and should provide navigation throughout the model through the use of hyperlinks.

7.1.4 Usability

First Contact

The users initial experience with the system should be one in which the task of initially constructing and modeling parts of the system is easy. This feature is analyzed based on the user's first experience with the tool without referring to any documentation.

Easy to Use

The tool should remain easy to use even when dealing with complex diagrams and objects. For this criterion the focus is the tool's ability to hide and reveal information allowing the user to focus on specific details.

Pick List

The modeling tool should provide pick lists in several key interfaces:
Collaboration and Sequence Diagrams - The tool should allow an object to be assigned to a class from a list of the classes in the model. It should allow the messages sent between objects to be chosen from a valid

list of methods for the object (class) which is receiving the message. The pick list feature contributes significantly to the intuitiveness of the modeling tool and may be considered a must-have feature. The development of sequence and collaboration diagrams is greatly facilitated by being able to quickly select the message you want to send from one object to another.

Interface Presentation

The interface should have a layout which is both consistent and aesthetically pleasing. This category typically examines areas such as fonts, labels on diagrams, facilities provided for viewing diagrams, and the visibility of different states of the model.

Documentation and Help File

Good documentation and search facilities help the user to learn the tool and feel comfortable around the tool. This determines how quickly a user can perform functions using the tool. Distinguishing the menus and the submenus makes readability easier. Name completion facilities, short-cut keys and learning aids can heavily influence the speed with which a user can perform common tasks.

7.1.5 Printing Support

The modeling tool should allow accurate renditions of large diagrams to be produced through multi-page printing. Print preview and scaling functionality should be supported to allow ease of fitting the diagram to the desired number of pages. The ability to fit a diagram to a single page is high on this list.

7.1.6 Exporting Diagrams

One key feature is the ability to export diagrams into a format that may be imported into a word processing document or a web page. The most popular graphics formats used for export are GIF, PNG and JPEG. When exporting, the tool should allow you to define the preferred resolution and size of the graphic that is produced.

XMI, an Object Management Group (OMG) standard, is an interchange format which has the potential to finally allow seamless sharing of models between development tools. For example, rather than writing scripts within a UML modeling tool to create reports, a user could simply export the model under development using XMI and import the model into a specialized report writing tool. It is therefore good that a tool should be able to export diagrams to XMI format.

7.1.7 Robustness

A UML tool should have rock-solid reliability and consistency. This is to prevent users from losing potentially hours of productivity, when the tool crashes in the middle of a design session, or corrupts a model which hasn't been backed up. A tool which causes hours of work to be lost due to a crash or file corruption is very unsatisfactory. A strategy to be considered here is to have the UML tool automatically to save a model in the background at periodic intervals.

7.1.8 New Release

The modelling tool selected should continue to be actively improved through bug fixes, performance improvements, and the addition of new features. Another factor is that these tools have to adapt to the new technological advancements in hardware and software environments. Since there is a big investment in time and money in a tool, it is not easy to change to another modelling tool. We can determine if a product is evolving by enquiring for a detailed schedule of recent releases and a roadmap for the product's future and by looking closely at the rate at which features and improvements have been made.

New versions and improvements in functionalities are good, provided the new version of the tool is backward compatible with the older versions. It is unreasonable that one should be stranded with old design diagrams which are not compatible with new versions of the tool. So it is necessary to consider this feature when deciding on a tool.

7.1.9 Round Trip Engineering

The ability to both forward and reverse engineer source code (Java, C++, and CORBA IDL) is a complex requirement that vendors support with varying degrees of success. The successful combination of these two features, forward and reverse engineering is defined as round-trip engineering. [1]

Code Generation or Forward Engineering

Code Generation is the process of generating code in the respective programming language for the components defined in the design. It is also possible to generate the code for the relationship between the components. Once this is generated the programmer just has to implement the logic of each operation.

Reverse Engineering

Reverse engineering is the ability of the tool to recognise the new classes, methods and attributes that the programmer adds to the application. This can be done during the development of the application or during maintenance once the application is deployed. Reverse engineering is very useful both to transform code into a model when no model previously existed, as well as to resynchronize a model with the code at the end of iteration.

During an iterative development cycle, once a model has been updated as part of the iteration, another round of forward engineering should allow code to be refreshed with any new classes, methods or attributes that have been added to the model. This step is less commonly adopted by developers because many tools can hopelessly mangle source code in the process. The problem is that the source code contains much more than the model; tools must be very adept at reconstructing the source code that existed prior to the new round of forward engineering.

At minimum, the modelling tool should successfully support forward engineering the first time and reverse engineering throughout the process. Also, the tool should have no trouble reverse engineering the full Java language. The way to verify this feature is to implement your own source code and try the round trip engineering on your code.

7.1.10 Data Modelling

The object modelling tool should allow integration with data modelling facilities. This can be done by

1. Allowing an object model to be transformed into DDL
2. Exporting metadata to a data modelling tool which can import the metadata and use it as the basis for a data model.

7.1.11 Model navigation

The modelling tool should provide strong navigational support to allow a developer to navigate through all the diagrams and classes in the model. A directory or pick list of classes sorted by name is one way to allow a designer to jump to the desired class on a diagram.

For large diagrams, the tool should provide ease of navigation when zooming and panning. The tool should also allow ease of navigating to the source code for a class when round-trip engineering is being used.

7.1.12 Diagram views

The modelling tool should facilitate customization of the view of a class and its details. For instance, it should be possible to exclude all get/set methods from the diagram since they tend to clutter, rather than clarify a diagram. The full signature of methods should be allowed to be shown or hidden easily, depending on the level of detail desired. The visibility of attributes and methods (private, protected, public) should be another dimension used to select what to show or hide on the diagram.

7.1.13 Platform

In order to maximize an investment in a modelling tool, one has to carefully consider the platforms on which the tool will run. Java's Swing user interface allows cross-platform graphical user interface (GUI). So if the UML tool is built using a cross platform GUI, one can look over the issue of platforms.

However, cross-platform tools need to be supported on common platforms such as Linux in order to achieve large-scale adoption by programmers. Sun had originally done little to promote Java on Linux. But recent industry initiatives, principally from IBM, which has pledged broad-based support for Linux on all of its hardware platforms and is supporting the Apache/Jakarta project, are now rapidly pushing Java onto Linux. Perhaps because IBM has moved to distribute its version of JDK 1.1.8 to the major Linux vendors, Sun has been compelled to support the distribution of a fully functional JDK 1.2 (Java 2, with Swing) for Linux. This Java port to Linux has been largely accomplished through the efforts of the Blackdown Group [15]. So a tool developed on a platform independent programming environment, gives a user the freedom to adopt any platform.

7.1.14 Multi-user Support

When working in a team oriented environment it is essential that the tool provides support for multiple users. This support is generally required in the form of multiple user access to the development software, which in turn requires users to be constrained by predefined permissions. The changes made by each user to a model should be backed up and made available to the next user.

7.1.15 UML Support

While many tools claim full support for UML 1.3, in reality this is a complex requirement and some tools may not live up to advertised claims for full support. At minimum, the diagrams which should be supported are the Use-case, Class, Collaboration, Sequence, Package, and State diagrams.

7.1.16 Support for Language

This feature deals with the different languages supported for code generation. Some of the common languages that are supported by tools include: JAVA, C++, VC++, CORBA, ADA, J2EE, and C #, Visual Basic.net, CORBA IDL and Visual Basic 6

7.1.17 Installation

Installation of any tool should be fairly easy and the tool should be up and running without many problems. One should be aware of all the prerequisites before considering a tool. Some tools require a database for use as a repository and others require an application server. It is essential to study the kind of application you develop before selecting a tool. Most of the prerequisites are open source software.

7.1.18 Class Diagram Features

- ***Class box size flexibility***
- ***Line flexibility***
- ***Independent placement of association end names***
- ***Preservation of position of end names and multiplicity labels***
- ***Moderate binding of relationship lines***
- ***Distinguish between remove from diagram and delete from model***
- ***Restoration of relationships in new diagrams***
- ***N-ary associations*****:** An N-ary association is an association among three or more classifiers (a single classifier may appear more than once). Each instance of the association is an n-tuple of values from the respective classifier. A binary association is a special case with its own notation.
- ***Undo functionality for diagrams***
- ***UML profiles (stereotypes and tagged values):*** A UML profile is made up of one or more “stereotypes” that may have “tagged values” and “constraints” [24]. Profiles are sometimes referred to as the ‘lightweight’ built-in extension mechanisms of UML, in contrast with the ‘heavyweight’ extensibility mechanism as defined by the Meta-Object Family (MOF) specification. This is because there are restrictions on how UML profiles can extend the UML metamodel [25]. These restrictions are intended to ensure that any extensions defined by a UML profile are purely additive [6].

7.2. Other Features

7.2.1 Sequence Diagram Features

UML sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are used to model the following:

1. ***Usage scenarios*****:** A usage scenario is a description of a potential way your system is used. This includes the basic course and the alternate course.
2. ***The logic of methods:*** Sequence diagrams can be used to explore

the logic of a complex operation, function, or procedure.

3. ***The logic of service:*** A service (web-services and business transactions) is effectively a high-level method, often one that can be invoked by a wide variety of clients. [17]

Sequence diagrams are the most popular UML artefacts for dynamic modelling, which focuses on identifying the behaviour within your system. Therefore it should be part of a UML tool.

7.2.2 Use-Case Diagram Features

Use-Case Diagrams can be used to describe the functionality of a system in a horizontal way. That is, rather than merely representing the details of individual features of your system, use-case diagrams can be used to show all of its available functionality. Use-Case diagrams have 4 major elements: The *actors* that the system you are describing interacts with, the *system* itself, the *use-cases*, or services, that the system knows how to perform, and the lines that represent *relationships* between these elements. [26, 6, 5]

Hence a tool should support the basic notations of a use-case diagram like actors, include, extends etc.

7.2.3 Support Robustness Diagrams

In this research project the ICONIX methodology is used as the process for designing. One of the steps in this process is to draw robustness diagrams. Some tools do not support the different notations in the robustness analysis. This may not be an essential feature but for companies that use these methodologies as one of their standards one should take this feature into consideration.

When considering this feature one should analyse the following: the tools support the notations used to represent the types of stereotypes, the rules and constraints applied while drawing the diagrams.

8. Case Study: Juke Box

The application chosen for the design is a fairly complex one, covering the prominent features of object oriented methodology like associations, generalisation, inheritance etc. The application chosen will be an automated jukebox, which incorporates user interfaces, storing, and retrieving data from a database, an administration interface, which allows future enhancements to the system, and also an interface with other systems like a cashbox.

The example chosen is to develop software for a digital jukebox. The logic of the overall system is summarised as follows:

> *Its external construction consists of a case containing a display screen with outputs to an audio amplifier and input from a track-ball. There is a cash box, which is a separate unit. It accepts, validates and counts coins but cannot give change. The music is stored in the database with all the details of track and its cost. The display screen displays the list of tracks and the trackball highlights one item from the list of tracks. A request for payment is made. Then the selected song is played. For administrative purposes the user should be allowed to enter data and produce usage reports. A detailed description of this example is given in the reference [12].*

This jukebox application will be taken through the complete process of an end–to–end application. A software application that goes through a whole Software Development Life Cycle (SDLC) process is called an end-to-end application.

9. Evaluation methodology

9.1 Evaluation Using Case Study

The case study described in section 8 will be used to do the evaluation. The example was studied. After laying down the requirements a design was developed. The same design was repeated on each tool. Hence this helps to understand and compare how each tool handles the development of the same application design. Then the code generation is done on each tool from the respective design. In this area we can compare the different tools by testing the different features in round trip engineering. Some of these include:

- → The code generation set up
- → The readability of the code generated by including comments.
- → Reverse engineering set up.
- → Feasibility to do reverse engineering.

In the code generated, the business logic is implemented for the operations and tested to produce a working prototype. The implementation is done in JAVA. The research concentrates more on the evaluation of the UML tools hence only part of the prototype is developed. Since the UML tool only generates a skeletal code for the user, the business logic plugged in by the user does not play a critical role in the evaluation.

9.2 Evaluation Process

The set of features was drawn up and defined. Each feature is defined clearly so as to eradicate duplication of features and definition of these features. It also clearly sets a boundary around each feature. Each feature can be further divided into sub features. For example consider the feature "round trip engineering". This can be subdivided as shown in Table 8.1

Features	***Sub-Features***
ROUND TRIP ENGINEERING	• Code Generation • Reverse Engineering • Synchronization With Editor

Table 8.1 Example of subdivision of a feature.

The evaluation is done using a "rating system" where a weighting is given for each feature and a rating is given based on the tool. The weighting for each sub-feature is fixed, based on the significance and the importance of each of the features when compared with the rest of the sub-features. This prevents some tools from scoring a higher value for a feature, which is not essential. If a feature is absolutely essential then it is assigned a 3. If the feature pleases the eye yet it is not significant in terms of production then it is given a 1. The weightings given to the features in this research are based on what was felt to be significant. This judgement was based on extensive literature survey, complaints from forums and discussion with users. The weightings are shown in Table 8.2

Nice to have Feature	Good Feature	Essential Feature
1	2	3

Table 8.2

The ratings for the features are given based on how well the functionality is implemented, for each tool chosen. If a feature is poorly implemented then it is assigned one but if it is implemented well then it gets a three. For example consider the feature "The class box flexibility". In some tools this feature is there yet it does not serve the purpose where, it should reveal all the attributes and methods on increasing the size. This is a poor implementation and it is assigned a one. Figure 8.3 shows the score for the ratings.

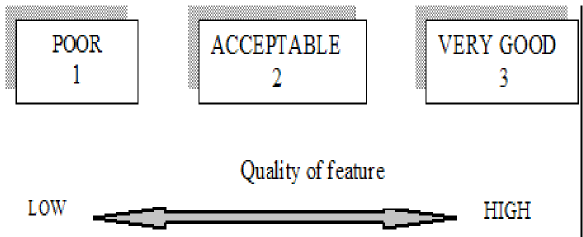


Figure 8.3

The product of the weighting and rating is calculated for each feature and the total sum of the products gives the final score for each tool. This can be summarised into the following formula:

Score for a tool =
$\sum$ (weighting of
each feature $\times$ rating of tool for
that feature)

9.3 Evaluation of Tools

The tools chosen for evaluation are ArgoUML v0.14, MasterCraft for JAVA v6.0, Rational Rose Enterprise Edition 2001A.04.00, Together Control Center v6.1. ArgoUML is an open source software while the other three are commercial softwares. MasterCraft is a UML based Component Modeller and it uses a Repository-driven development process [19]. Rational Rose supports two elements of modern software engineering: component based development and controlled iterative development [20]. Together is another product which is as popular in industry as Rational Rose. In all these modeling tools code generation can be done in JAVA.

The evaluations for each tool are shown below. The weightage is given to prioritize the features. The values are given based on the author's experiences with the tools. The matrix given in Table 9.1 could be used as an evaluation tool to compare any set of UML tool. The tool has the following headers: Features, Weightings and list of UML Tools used for comparison. The Features listed are indispensable in any UML tool. The weightings given for each feature could be prioritised according to the user's priority list. And finally a set of UML tools can be used for comparison. This can be done by plugging in quantitative values and calculating the highest score.

Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
Repository:					
database	**2**	1	2	1	3
Customization:					
Components and tools	**2**	3	1	2	1
HTML documentation:					
Generate Web Reports:	**3**	2	1	3	2
Save diagrams to include in reports	**3**	3	1	3	3
Usability:					
First Contact:	**2**	3	3	2	1
Ease of use:	**3**	2	3	3	2
Pick List:					
Pick list of the classes in the model while drawing.	**2**	3	1	1	1
Tree structure in a pane where all the classes can be viewed.	**3**	3	3	3	3
Select methods to draw sequence or collaboration diagram.	**3**	2	1	3	3
Drag and drop	**3**	3	3	3	3

Features	*Weightings*	*Rational Rose*	*Argo UML*	*Together*	*Master Craft*
Interface Presentation					
Allows changing the font and size of labels on diagrams.	**3**	3	1	3	3
States are visibly distinct	**2**	3	2	2	2
Documentation and help files					
step by step documentation	**3**	3	2	3	2
Online help	**2**	3	1	3	1
Keyword search facilities	**3**	1	1	1	1
Short – cut keys	**1**	1	1	1	1
Printing Support:					
Fit diagram to a single page	**3**	3	3	3	2
Print preview	**3**	3	1	3	3
Scaling functionality	**3**	2	1	3	1
Exporting Diagrams:					
Save diagrams any picture editor format:	**3**	3	1	3	3
XMI format:	**2**	3	3	3	3
Robustness:					

Features	*Weightings*	*Rational Rose*	*Argo UML*	*Together*	*Master Craft*
Does tool crash or corrupt diagrams:	**3**	2	3	3	3
Automatic saving at periodic intervals:	**1**	1	1	3	1
New Releases:					
Any New version releases announced:	**3**	3	1	2	3
Round Trip Engineering:					
Code generation:	**3**	3	3	3	3
Reverse engineering:	**3**	3	2	3	
Synchronization with editor:	**2**	1	3	3	2
Model navigation:					
Zooming and panning:	**3**	3	1	3	3
Navigating between source code and diagram:	**2**	1	2	3	
Diagram views:					
Customizing details of classes:	**3**	3	3	2	2
Reveal and hide methods:	**3**	2	1	2	2
Visibility of methods and attributes:	**3**	3	3	3	3

Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
UML support:					
Key Notation parts					
Use-case diagrams	**3**	2	3	3	3
Class diagrams	**3**	3	3	3	3
Sequence diagrams	**3**	3	1	3	3
Collaboration diagrams	**3**	3	3	3	3
State diagrams	**3**	3	3	3	3
Activity diagrams	**3**	3	3	3	3
Component diagrams	**3**	1	1	3	1
Deployment diagrams	**3**	1	3	3	3
Package diagrams	**3**	3	1	1	3
Class box size flexibility	**3**	3	3	2	2
Line flexibility	**2**	3	3	2	2
Independent placement of association end names	**3**	3	3	3	3
Independent placement of multiplicity labels	**3**	3	3	3	3
Preservation of position of end names and multiplicity labels	**3**	3	1	3	3
Distinguish between remove from diagram and delete from	**3**	3	3	1	3

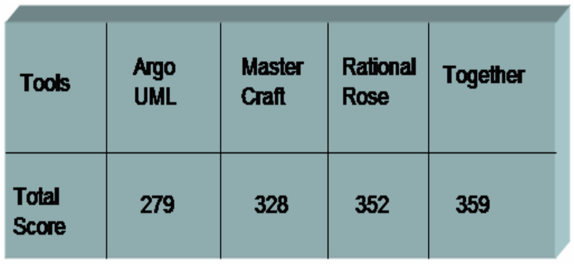
Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
model					
Restoration of relationships in new diagrams	**3**	3	1	3	2
N-ary associations	**3**	3	3	3	3
undo functionality for diagrams	**2**	1	1	3	3
UML profiles (stereotypes and tagged values)	**2**	3	1	2	2
Total Score		***352***	***279***	***359***	***328***

Table 9.1 Evaluation of tools with ratings

10. Conclusions and Observations

The evaluation technique provides an efficient means of isolating both the positive and negative features of UML tools. From a business point of view cost and time are some of the major concerns in a software industry. Tools do play a role in the development of cost effective software. Any user should feel comfortable and confident in the tool he uses. Above all, tools are used to make the job easier. All these aspects make it worthwhile to invest time and effort to assess the tools before choosing the right tool.

The four tools evaluated and rated were ArgoUML, Master Craft, Rational Rose, and Together. The table 10.1 presents a summary of the final score obtained for each tool.



Tools	Argo UML	Master Craft	Rational Rose	Together
Total Score	279	328	352	359

Table 10.1

These tools are evaluated considering

only the technical aspects and they involve the author's experience with the tools, while designing and implementing the jukebox example. The ratings and weighting of these tools are also judged based on the author's findings on these tools. Discussions with users, comments in forums have also been a factor in studying and coming to conclusion on these tools. Non- technical aspects are not part of the evaluation tool.

The matrix given in Table 9.1 could be used as an evaluation tool to compare any set of UML tool. The tool has the following headers: Features, Weightings and list of UML Tools used for comparisons. The Features listed are indispensable in any UML tool. The weightings given for each feature could be prioritised according to the user's priority list. And finally a set of UML tools can be used for comparison. This can be done by plugging in quantitative values and calculating the highest score.

11. Future Improvements

Principles of Software, though new to the industry has improved and advanced immeasurably. Software is so complex and as it grows it tends to be even more complex and difficult to maintain.

The evaluation criteria can be modified further. Software is now built at multi-site and with multiple users. Issues like authentication, change control, robustness of saving data, version control etc. are important sub-features in this feature. These could be investigated further with the example implemented. Only part of the example has been implemented in this research due to time constraints. This example could be implemented further to be a complete example.

The XML Metadata Interchange Format [XMI] standard from the Object Management Group (OMG) [20], is one of the most exciting recent developments in the UML developer community. XMI is an interchange format, which has the potential to finally allow seamless sharing of models between best-of-breed development tools [15]. XMI uses XML to represent model information. During iterative development of a model, it is very productive to look at the UML diagram and matching source code in adjacent windows. When your project starts to mature, it may be good to have access to the metrics for your model. These are some of the features that vendors are trying to incorporate into their product and should indubitably be part of the evaluation criteria.

12. References

[1] Comparison of UML Modeling Tools: Enterprise Architect and Rational Rose, http://consulting.dthomas.co.uk/ooad_articles_resources/Comparison_of_UML_Modelling_Tools.pdf last accessed on 19 October 2003.

[2] Vliet, H.V. Software Engineering Principles and practices, ISBN 0-471-97508-7, John Wily & Sons Ltd, New York, 2000.

[3] The Systems Development Life Cycle, http://www.muskalipman.com/VBObjects/SDLC.pdf last accessed on 11 December 2003.

[4] Software Design Principles, http://cs.wwc.edu/~aabyan/Design/softwareDesign.html last accessed on 29 November 2003.

[5] Introduction to OMG's Unified Modelling Language™ (UML™), http://www.omg.org/gettingstarted/what_is_uml.htm last accessed on 19 October 2003.

[6] OMG Unified Modeling Language Specification, http://www.omg.org/docs/formal/01-09-67.pdf last accessed on 19 October 2003.

[7] UML- a tutorial, http://ranger.uta.edu/~alp/cse5330/UML/uml_tutorial.pdf last accessed on 21 October 2003.

[8] UML tutorial, http://www.geocities.com/sriskanthaverlk/UML_tutorial.pdf last accessed on 21 October 2003.

[9] Deitel, H. M., Deitel P. J., JAVA How To Program, ISBN 0-13-0345151-7 Prentice Hall, New Jersey, 2001.

[10] Iconix Process, http://c2.com/cgi-bin/wiki?IconixProcess last accessed on 4 November 2003.

[11] The ICONIX Process http://dept.ee.wits.ac.za/~slevitt/elen533/ICONIX.pdf last accessed on 4 November 2003.

[12] Prof. B. Dwolatzky, Notes on Software Design Process, University of Witwatersrand, 2003

[13] Advanced Tools for UML: now and in the future, http://www.dcs.ed.ac.uk/home/pxs/uml2000.pdf last accessed on 21 October 2003.

[14] Building a UML Editing Tool, http://216.239.57.104/search?q=cache:k7OJj5WZU-IJ:www.itee.uq.edu.au/~comp6801/project-components/BrettCampbell.pdf++OMG%22UML+Tools%22+%22Evaluation+Criteria+%22&hl=en&ie=UTF-8 last accessed on 19 October 2003.

[15] Objects by Design: Choosing a UML Modelling Tool, http://www.objectsbydesign.com/tools/modeling_tools.html last accessed on 21 October 2003.

[16] Chapter 4: Software: Systems and Application Software http://66.102.11.104/search?q=cache:5eYY16G7pl8J:www.csus.edu/indiv/v/velianitis/175/Chapter4Software.ppt+%22types+of+software%22+like+open+source,+off+the+shelf&hl=en&ie=UTF-8 last accessed on 16 March 2004

[17] UML 2 Sequence Diagram Overview http://www.agilemodeling.com/artifacts/sequenceDiagram.htm last accessed on 17 March 2004

[18] All about Sequence Diagrammes http://www.lindesay.co.nz/manuals/SequenceSketcherHelp/whatare.html last accessed on 17 March 2004.

[19] MasterCraft java , http://www.tatamastercraft.com/downloads/JavaFlier.pdf last accessed on 19-Feb-2004.

[20] Case: Rational Rose 81940 A Seminar on Reverse Engineering, Software Systems Laboratory, Tampere University of Technology Fall, 2000, http://www.cs.tut.fi/~tsysta/sem/Reports/Rose.pdf last accessed on 20 March 2004.

[21] Software Development, http://www-306.ibm.com/software/awdtools/library/standards/ last accessed on 19 March 2004.

[22] Martin J., Odell J. J., Object Oriented analysis and design, ISBN: 0-13-630245-9, PTR Prentice Hall, New Jersey, 1992.

[23] Use-case Driven Object Modeling with UML http://pst.web.cern.ch/PST/HandBookWorkBook/Handbook/SoftwareEngineering/UCDOM_summary.html last accessed on 19 March 2004.

[24] Enterprise Architect 3.10 - UML Profile XML File Format Information http://www.sparxsystems.com.au/profile/UML_Profile_Information.pdf last accessed on 20 March 2004.

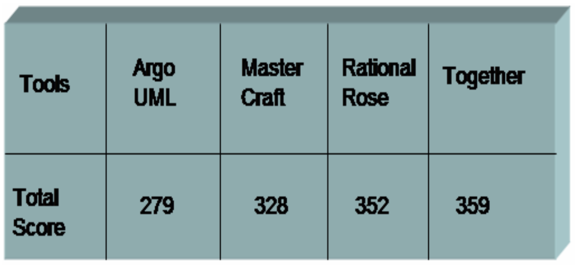
[25] Comparing UML 1.4/1.5 and 2.0 http://66.102.11.104/search?q=cache:2cfOUiO-O_YJ:cs.ua.edu/630/Notes/2003-02-06/UML%2520Presentation%2520Note.doc+Profiles+are+sometimes+referred+to+as+the+lightweight&hl=en&ie=UTF-8 last accessed on 19 March 2004.

[26] UML Use-case Diagrams: Tips and FAQ http://www.andrew.cmu.edu/course/90-754/umlucdfaq.html last accessed on 19 March 2004.

DISCUSSION AND CONCLUSION

The evaluation technique in this research work provides an efficient means of isolating both the positive and negative features of UML tools. From a business point of view cost and time are some of the major concerns in a software industry. From the early ages, tools have always played a role in aiding man to get his work done smoothly and efficiently. Tools do play a role in the development of cost effective software. Any user should feel comfortable and confident in the tool he uses. Above all, tools are used to make the job easier. All these aspects make it worthwhile to invest time and effort to assess the tools before choosing the right tool.

The four tools evaluated and rated in this research work were ArgoUML, Master Craft, Rational Rose, and Together. Though ArgoUML has the least score it is open source software while other tools come with a high cost. The table 3 draws up a summary of the final score obtained for each tool.



Tools	Argo UML	Master Craft	Rational Rose	Together
Total Score	279	328	352	359

Table 3 Total score from the evaluation of tools

These tools are evaluated based only on the technical aspects. The value given for each feature and tool is based on my experience with the tools, while designing and implementing the jukebox example. The ratings and weighting of these tools are also judged based on my findings in these tools. Discussions with users, comments in forums have also been a factor in studying and coming to conclusion on these tools. The evaluation tool emphasizes on the technical features in a UML tool. These technical features are mainly functional and usability issues. Non-technical aspects such as cost, training, local support, availability of resource and developmental environment, are not part of the comparison matrix because it is beyond the scope of this project report.

This research report is unique by presenting an evaluation tool / matrix [Appendix VI] to compare any set of UML tool. The tool has the following headers: Features, Weightings, UML Tools used for comparisons. The Features listed are indispensable in any UML tool. The weightings given for each feature could be customised according to the user's priority list. And finally a set of UML tools can be used for comparison. This can be done by plugging in quantitative values and calculating the highest score.

Software, though new to the industry has improved and advanced immeasurably. Software is so complex and as it grows it tends to be even more complex and difficult to maintain. Hence it was realised that there was a crisis in software. Acquiring all the user requirements is an important factor. Yet one has to capture all these requirements in the design. Further your design should cater for possible, future improvements in the system. Hence your design is not a fixed blue print of your system but it should be improved and maintained with new enhancements. This maintenance has to be done even after the original programmers of the system have moved to the next project.

The evaluation criteria can be modified further. Software is now built at multi-site and with multiple users. Issues like authentication, change control, robustness of saving data, version control etc. are important sub-features in this feature. These could be investigated further with the example implemented. Only part of the example has been implemented in this research due to time constraints. This example could be implemented further to be complete example.

The XMI standard from the Object Management Group (OMG) is one of the most exciting recent developments in the UML developer community. XMI is an interchange format which has the potential to finally allow seamless sharing of models between best-of-breed development tools [15]. During iterative development of a model, it is very productive to look at the UML diagram and matching source code in adjacent windows. When your project starts to mature, it may be good to have access to the metrics for your model. These are some of the features that vendors are trying to incorporate into their product and should indubitably be part of the evaluation criteria.

Other new features that could be included in the tools could be

- ***Management tools***: A nice feature that should be integrated with modelling tools is the ability to export modelling information into a tool that will allow you to track the progress of both the design and implementation of your project.

- ***Auto generation of interaction and state diagrams:*** This function is the ability for modelling tools to help in the generation of interaction and state diagrams. This could be achieved using a trace file. Once the trace file is created, the modelling tool would be used to analyze the trace in order to find the patterns of object interactions.

REFERENCES \BIBLIOGRAPHY

[1] Comparison of UML Modelling Tools: Enterprise Architect and Rational Rose, http://consulting.dthomas.co.uk/ooad_articles_resources/Comparison_of_UML_Modelling_Tools.pdf last accessed on 19 October 2003.

[2] Vliet, H.V. Software Engineering Principles and practices, ISBN 0-471-97508-7, John Wily & Sons Ltd, New York, 2000.

[3] The Systems Development Life Cycle, http://www.muskalipman.com/VBObjects/SDLC.pdf last accessed on 11 December 2003.

[4] Software Design Principles, http://cs.wwc.edu/~aabyan/Design/softwareDesign.html last accessed on 29 November 2003.

[5] Introduction to OMG's Unified Modelling Language™ (UML™), http://www.omg.org/gettingstarted/what_is_uml.htm last accessed on 19 October 2003.

[6] OMG Unified Modelling Language Specification, http://www.omg.org/docs/formal/01-09-67.pdf last accessed on 19 October 2003.

[7] UML- a tutorial,
http://ranger.uta.edu/~alp/cse5330/UML/uml_tutorial.pdf last accessed on 21 October 2003.

[8] UML tutorial,
http://www.geocities.com/sriskanthaverlk/UML_tutorial.pdf last accessed on 21 October 2003.

[9] Deitel, H. M., Deitel P. J., JAVA How To Program, ISBN 0-13-0345151-7 Prentice Hall, New Jersey, 2001.

[10] Iconix Process, http://c2.com/cgi-bin/wiki?IconixProcess last accessed on 4 November 2003.

[11] The ICONIX Process
http://dept.ee.wits.ac.za/~slevitt/elen533/ICONIX.pdf last accessed on 4 November 2003.

[12] Prof. B. Dwolatzky, Notes on Software Design Process, University of Witwatersrand, 2003

[13] Advanced Tools for UML: now and in the future,
http://www.dcs.ed.ac.uk/home/pxs/uml2000.pdf last accessed on 21 October 2003.

[14] Building a UML Editing Tool, http://216.239.57.104/search?q=cache:k7OJj5WZU-IJ:www.itee.uq.edu.au/~comp6801/project-components/BrettCampbell.pdf++OMG%22UML+Tools%22+%22Evaluation+Criteria+%22&hl=en&ie=UTF-8 last accessed on 19 October 2003.

[15] Objects by Design: Choosing a UML Modelling Tool, http://www.objectsbydesign.com/tools/modeling_tools.html last accessed on 21 October 2003.

[16] Introduction to ArgoUML, http://216.239.59.104/search?q=cache:QAQHdNCZruMJ:www-serl.cs.colorado.edu/downloads/serl-talks/2002.02.11-ArgoUML.pdf+ARGOUML+features&hl=en&ie=UTF-8 last accessed on 22-Feb-2004

[17] ArgoUML User Manual, A tutorial and reference description of ArgoUML, http://www.ime.montana.edu/~emooney/ime534/ArgoUML-docs-0.10/manual/index.html last accessed on 22-Feb-2004.

[18] Case Tools Compared Crisis Management, Frank Haubenschild http://www.linux-magazine.com/issue/11/CASE_Tools.pdf last accessed on 22-Feb-2004.

[19] Using Rose Rational Rose®, http://aris.fri.uni-lj.si/~damjan/RIS_vaje/Rose_usingrose.pdf last accessed on 22-Feb-2004.

[20] Rational Rose, http://c2.com/cgi/wiki?RationalRose , last accessed on 19-Feb-2004.

[21] MasterCraft: the Lifecycle Development Tool http://www.omg.org/mda/mda_files/MasterCraftMDAV.pdf, last accessed on 19-Feb-2004.

[22] JAVA 2 Complete , Sybex San Francisco, ISBN - 0-7821-2468-2 ,1999,

[23] CDROM, Inside unified Modelling language, Rational software Corporation 2002.

[24] Eichelberger H., Nice Class Diagrams Admit Good Design , Software Visualization Proceedings of the 2003 ACM symposium on Software visualization, http://delivery.acm.org/10.1145/780000/774857/p159-eichelberger.pdf?key1=774857&key2=3665077701&coll=GUIDE&dl=ACM&CFID=14628654&CFTOKEN=73897503
last accessed on the 23-Feb-2004.

[25] Tutorial: Importing and exporting data from a text, filehttp://info.borland.com/techpubs/jbuilder/jbuilder9_bea/database/textimportexport/textfileimportexport_tutorial.html
last accessed on the 20-Feb-2004.

[26] 5 CASE Tools, http://uml.tutorials.trireme.com/uml_tutorial_5.htm last accessed on the 20-Feb-2004.

[27] MasterCraft java , http://www.tatamastercraft.com/downloads/JavaFlier.pdf last accessed on 19-Feb-2004.

[28] Tutorial MasterCraft Enterprise 6.0 JAVA for Windows, Tata Consultancy Services, Pune .

[29] Grand M., Knudsen J., JAVA Fundamental Classes, O'Reilly & Associations, ISBN 1-56592-241-7, 1997.

[30] Bishop J., Java Gently, Third edition, Addison Wesley, ISBN - 0 201 71050 1 , 2001.

[31] Objects by Design: Choosing a UML Modelling Tool, http://www.objectsbydesign.com/tools/modeling_tools.html last accessed on 21 October 2003.

[32] Chapter 4: Software: Systems and Application Software http://66.102.11.104/search?q=cache:5eYY16G7pl8J:www.csus.edu/indiv/v/velianitis/175/Chapter4Software.ppt+%22types+of+software%22+like+open+source,+off+the+shelf&hl=en&ie=UTF-8 last accessed on 16 March 2004

[33] UML 2 Sequence Diagram Overview http://www.agilemodeling.com/artifacts/sequenceDiagram.htm last accessed on 17 March 2004

[34] All about Sequence Diagrams
http://www.lindesay.co.nz/manuals/SequenceSketcherHelp/whatare.html last accessed on 17 March 2004.

[35] Fowler M., UML Distilled- A Brief guide to standard object modelling language, Third Edition, ISBN –032- 19368-7, Pearson Edition , 2004.

[36] Case: Rational Rose 81940 A Seminar on Reverse Engineering, Software Systems Laboratory, Tampere University of Technology Fall, 2000, http://www.cs.tut.fi/~tsysta/sem/Reports/Rose.pdf last accessed on 20 March 2004.

[37] Software Development,
http://www-306.ibm.com/software/awdtools/library/standards/ last accessed on 19 March 2004.

[38] Martin J., Odell J. J., Object Oriented analysis and design, ISBN: 0-13-630245-9, PTR Prentice Hall, New Jersey, 1992.

[39] Cook S., David J., Designing Object Systems – Object Oriented Modelling with Syntropy, ISBN: 1-13-203860-9, Prentice Hall, 1994.

[40] Enterprise Architect 3.10 - UML Profile XML File Format Information
http://www.sparxsystems.com.au/profile/UML_Profile_Information.pdf last accessed on 20 March 2004.

[41] Comparing UML 1.4/1.5 and 2.0
http://66.102.11.104/search?q=cache:2cfOUiO-O_YJ:cs.ua.edu/630/Notes/2003-02-06/UML%2520Presentation%2520Note.doc+Profiles+are+sometimes+referred+to+as+the+lightweight&hl=en&ie=UTF-8
last accessed on 19 March 2004.

[42] Use-case Driven Object Modelling with UML
http://pst.web.cern.ch/PST/HandBookWorkBook/Handbook/SoftwareEngineering/UCDOM_summary.html last accessed on 19 March 2004.

[43] User Guide for Together Control Center™ and Together® Solo, Downloadable at http://www.togethersoft.com/download/license.do or Together Control Center installation CDROM Version 6.1, last accessed on 01 April 2004.

[44] Accelerate Your Productivity, Borland® Together® Control Center® Technical Overview, http://www.borland.com/together/pdf/tgr61_controlcenter_techview.pdf, last accessed on 21 February 2004.

[45] Accelerate Your Productivity, Borland® Together® Control Center® Features Overview, http://www.borland.com/together/pdf/tgr61_controlcenter_datasheet.pdf, last accessed on 21 February 2004.

[46] MasterCraft Enterprise 6.0 JAVA Roles Overview, MasterCraft Group, TATA Consultancy Services, Pune.

[47] User Manual, MasterCraft Enterprise 6.0 JAVA for windows, MasterCraft Group, TATA Consultancy Services, Pune.

[48] Training Manual, MasterCraft Enterprise 6.0 JAVA for windows, MasterCraft Group, TATA Consultancy Services, Pune.

[49] Introduction to UML 1.3, MasterCraft Group, TATA Consultancy Services, Pune.

[50] UML Case Study 2, MasterCraft Group, TATA Consultancy Services, Pune.

[51] MasterCraft Enterprise 6.0 JAVA Overview, MasterCraft Group, TATA Consultancy Services, Pune.

[52] MasterCraft Enterprise 6.0 JAVA Analysis Overview_ 1 , MasterCraft Group, TATA Consultancy Services, Pune.

[53] MasterCraft Enterprise 6.0 JAVA Component Overview, MasterCraft Group, TATA Consultancy Services, Pune.

[54] MasterCraft Enterprise 6.0 JAVA Construction Overview _1, MasterCraft Group, TATA Consultancy Services, Pune.

[55] MasterCraft Enterprise 6.0 JAVA Construction Overview _2, MasterCraft Group, TATA Consultancy Services, Pune.

[56] MasterCraft Enterprise 6.0 JAVA Design Overview _1, MasterCraft Group, TATA Consultancy Services, Pune.

[57] MasterCraft Enterprise 6.0 JAVA Design Overview _2, MasterCraft Group, TATA Consultancy Services, Pune.

[60] MasterCraft Enterprise 6.0 JAVA GUIMod Overview _1, MasterCraft Group, TATA Consultancy Services, Pune.

[61] MasterCraft Enterprise 6.0 JAVA GUIMod Overview _2, MasterCraft Group, TATA Consultancy Services, Pune.

[62] MasterCraft Enterprise 6.0 JAVA JAVA_1, MasterCraft Group, TATA Consultancy Services, Pune.

[63] MasterCraft Enterprise 6.0 JAVA JAVA_2, MasterCraft Group, TATA Consultancy Services, Pune.

[64] MasterCraft Enterprise 6.0 JAVA Q++_1, MasterCraft Group, TATA Consultancy Services, Pune.

[65] MasterCraft Enterprise 6.0 JAVA Q++_2, MasterCraft Group, TATA Consultancy Services, Pune.

[66] MasterCraft Enterprise 6.0 JAVA Version Control Overview, MasterCraft Group, TATA Consultancy Services, Pune.

[65] MasterCraft Enterprise 6.0 JAVA Release Overview, MasterCraft Group, TATA Consultancy Services, Pune.

[66] MCE Java 6.0 Help, MasterCraft CDROM, MasterCraft Group, TATA Consultancy Services, Pune.

[67] Programmer's Guide, Q++ for MasterCraft Enterprise 6.0 JAVA for windows, MasterCraft Group, TATA Consultancy Services, Pune.

[68] Martin R.C., UML Tutorial: Part1 – A Class Diagrams last accessed on 11 – Dec – 2003.

[69] Perdita S., Small-Scale XMI Programming: A Revolution in UML Tool Use? http://www.dcs.ed.ac.uk/home/pxs/xmiImpact.pdf last accessed on 11 – Dec – 2003.

[70] Developing database applications, Borland JBuilder, http://ftp.vc-graz.ac.at/ftp.tu-graz.ac.at/jbuilder/techpubs/jbuilder9/database.pdf last accessed on 11 – Dec – 2003.

[71] The Java Tutorial, http://java.sun.com/docs/books/tutorial/ last accessed on 16– Apr–2003.

[72] Eckels B., Thinking in Java Second ed. Revision 12, Prentice Hall, New Jersey, ISBN: 0-13-027363-5, 2000.

APPENDIX I

END-TO-END APPLICATION

A CASE STUDY

Table of Contents

1.1 Introduction

A fairly complex system was chosen for the implementation and study of the UML tools. This section tries to define an end-to-end application. Then it takes the reader through the techniques and steps for designing the system providing sufficient examples. A brief description of the ICONIX process is given.

1.2 End-to-End Application

When building a bridge, the construction group does not start , by piling up bricks. Rather the requirements are analysed by taking into account factors like purpose of the bridge, type of transport it accommodates, finances, environment, expansion that could be made in the future to the bridge. The architect puts these factors into his design while designing. This design is analysed and agreed upon before the construction begins.

Constructing large, growing software systems is very similar to building a bridge yet with its own uniqueness. The problem to solve is first analysed and the requirements are defined in a very precise way. Then a design is made based on these requirements. Finally the construction process is started. There are a distinguishable number of phases in the development of software. The different phases in an SDLC are Requirements Engineering, Design, Implementation, Testing and Maintenance [2].

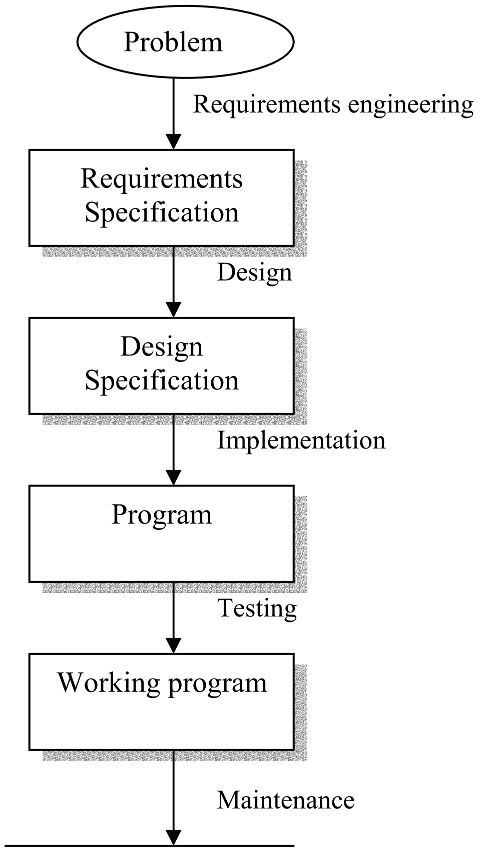


Figure 1.1 A simple of view of a software development life cycle [2]

We define an end-to-end process to be a complete Software Development Life Cycle (SDLC) process. And an application developed by stepping through all the phases in the development of software can be defined as an end-to-end application.

1.3 Application Chosen for Study

1.3.1 Description

The example chosen is to develop software for a digital jukebox. This example deals with a user interface a specialised hardware. Its external construction consists of a case containing a display screen with outputs to an audio amplifier and input from a track-ball.
There is a cash box which is a separate unit. It accepts, validates and counts coins but cannot give change. The music is stored in the database with all the details of track and its cost. The display screen displays the list of tracks and the trackball highlights the one item from the list of tracks. A request for payment is made. Then the selected song is played. For administrative purposes the user should be allowed to enter data and produce usage reports. A detailed description of this example is given in the reference [12].

The jukebox system was developed using the object oriented technology. The Unified Modelling Language, an OMG standard since1997, is a visualizing language for the modelling and development of software systems [5, 6]. The UML language has a set of diagram types and notations to represent the different concepts of object oriented programming. The main nine types of diagrams are: Class Diagram, Component Diagram, Deployment diagram, Use-case Diagram, Sequence Diagram, Activity Diagram, Collaboration Diagram, Statechart Diagram, Package diagrams. The notion used to represent concepts like generalisation, aggregation, inheritance etc. are explained in detail (see reference [6]).

1.3.2 Design

1.3.2.1 ICONIX Process

Unified modelling language, on its own is not sufficient to be used for designing of software. UML by itself is just a modelling language with notations and representation. But these notation and techniques can be optimised by using methodologies, tools, processes and guide lines for the design. The process used for this design is ICONIX process. This methodology was created by Doug Rosenberg. ICONIX uses robustness analysis as a bridge between use-cases (or Domain Model) and the code [10]. This methodology assumes an Object Oriented decomposition of the domain and it is use-case driven.

The diagram [Fig 1.2] below explains the ICONIX process. It consists of two parts: dynamic and static. The dynamic part includes the use-case diagram, robustness diagram and the sequence diagram. [11, 12]

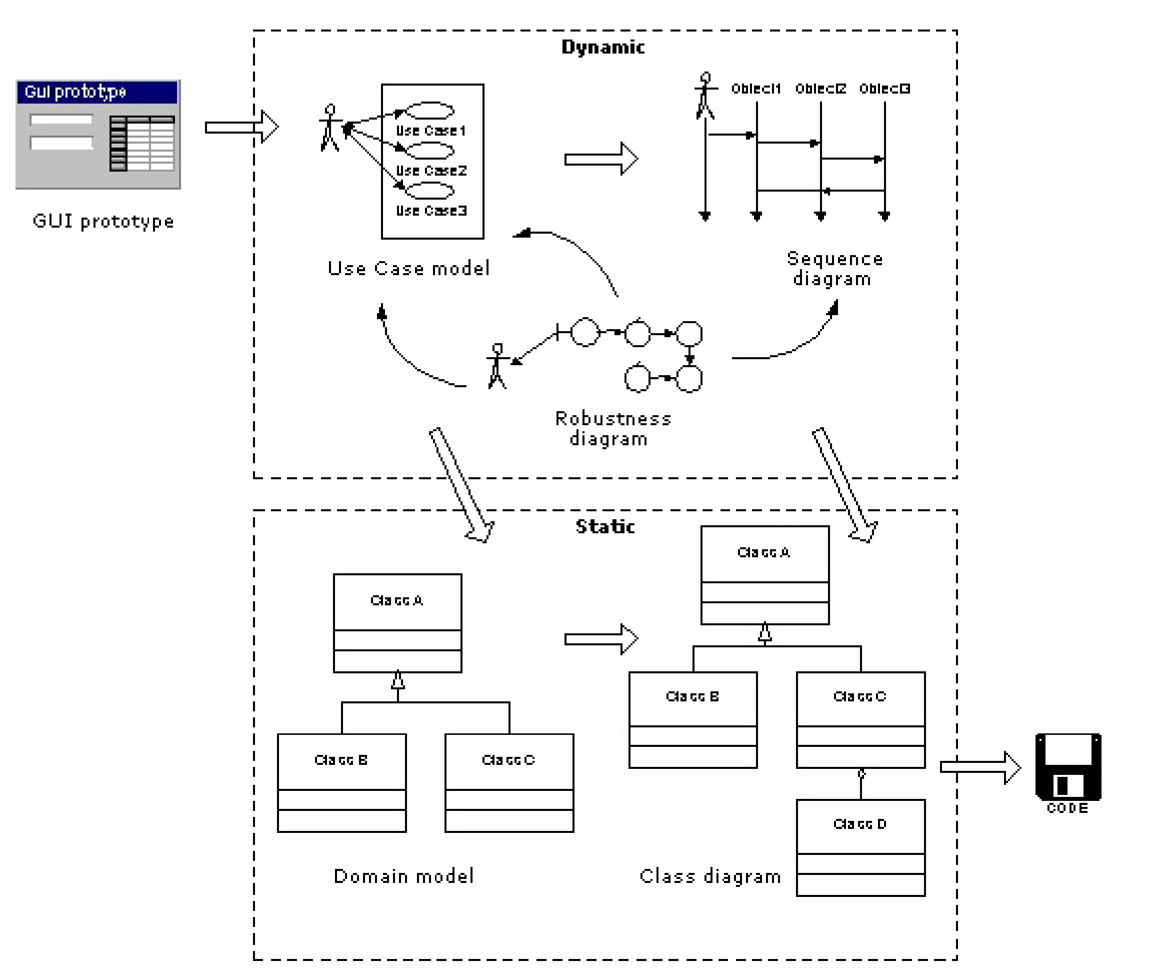


Figure 1.2 The ICONIX process [42]

Apart from guiding a designer through a sequence of diagrams the ICONIX process also contains a number of mile stones, which are steps at which the progress is reviewed. The different steps in the ICONIX process are:

Step 1: Informal statement of GUI & Prototype GUI
Step 2: Domain Modelling
Step 3: Use-case Modelling
Step 4: Requirements Review
Step 5: Robustness Analysis

Step 6: Preliminary Design Review
Step 7: Sequence Diagram
Step 8: Class Diagram
Step 9: Critical Design Review
Step10: Delivery

1.3.2.2 Case Study – Jukebox

Note: For detailed information on the design see reference [12].

Case Study: Informal statement of Problem & Prototype GUI

In the case study the Jukebox example is used along with the ICONIX process to bring forth a design for the application. Due to time constraints part of the design is implemented in an object oriented language JAVA. In the explanation given below examples are shown for each step to give a better understanding of how each step is achieved.

The example chosen for the study is a standard application for the evaluation. This application is to develop software for a jukebox.

Informal Statement

An equipment supplier to the entertainment industry has hired you to develop software for a digital jukebox. They have developed the specialised hardware. Its external construction consists of a very robust case containing a 14" display screen (like a PC monitor), outputs to an audio amplifier, a tracker-ball input device with a single select button, and a cash box.

a) The cash box is a separate unit which accepts, validates and counts coins. It does not give change. The control computer communicates with the cash box via an RS232 (serial) connection. It sets a counter in the cash box to an amount, in cents, corresponding to the required payment. As coins are deposited the counter is decremented until it reaches zero. An "okay" signal is then sent back to the control computer.

b) The music is stored on a database. Each track has a title, an artist, a playing time, and a cost (Le. different amounts might be charged for different tracks).

c) The display screen is primarily used to list the available tracks. The tracker ball and select button are used to highlight an item on the list and request that it is are played. The screen then displays the amount to pay. When the juke box is "waiting" and while it is playing a track, graphical images can be displayed on the screen.

d) For set up and administrative purposes a keyboard and printer can be connected to the unit to enter data and produce usage reports.

In discussing the requirements with you the equipment supplier says that he might, in the future, require the digital jukebox to allow the user to view videos or web pages. He would also like to use credit cards rather than cash for payment.

Figure 1.3 Informal statement of the problem domain [12]

A prototype of each of the screens in the system is drawn. For example a throw away prototype of the login GUI is shown below.

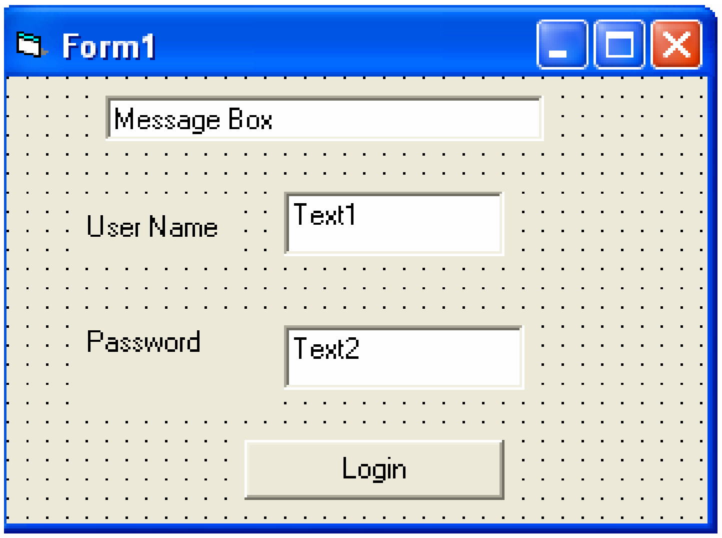


Figure 1.4 Prototype GUI of login

Case Study: Domain Modelling

In the domain model we try to identify the abstractions in the real world. These will include the main problem space conceptual objects that are going to participate in the system. You pick out all the nouns and the noun phrases from the informal statement. This is you candidate class. The next step is to sift through the candidate classes and eliminate items that are unnecessary or redundant. The last step is to show the relationship like, generalisation and aggregation, between these classes.

The sifted classes are shown below (Table 1.1):

Design Jukebox	Display Screen	Audio amplifier
Track Ball	Select button	Cash box
Ok Signal	Title	Required payment
Track	Graphical images	Database
Playtime	Web pages	Artist
Printer	Usage reports	List of tracks
Video	Credit cards	keyboard

Table 1.1 List of sifted classes

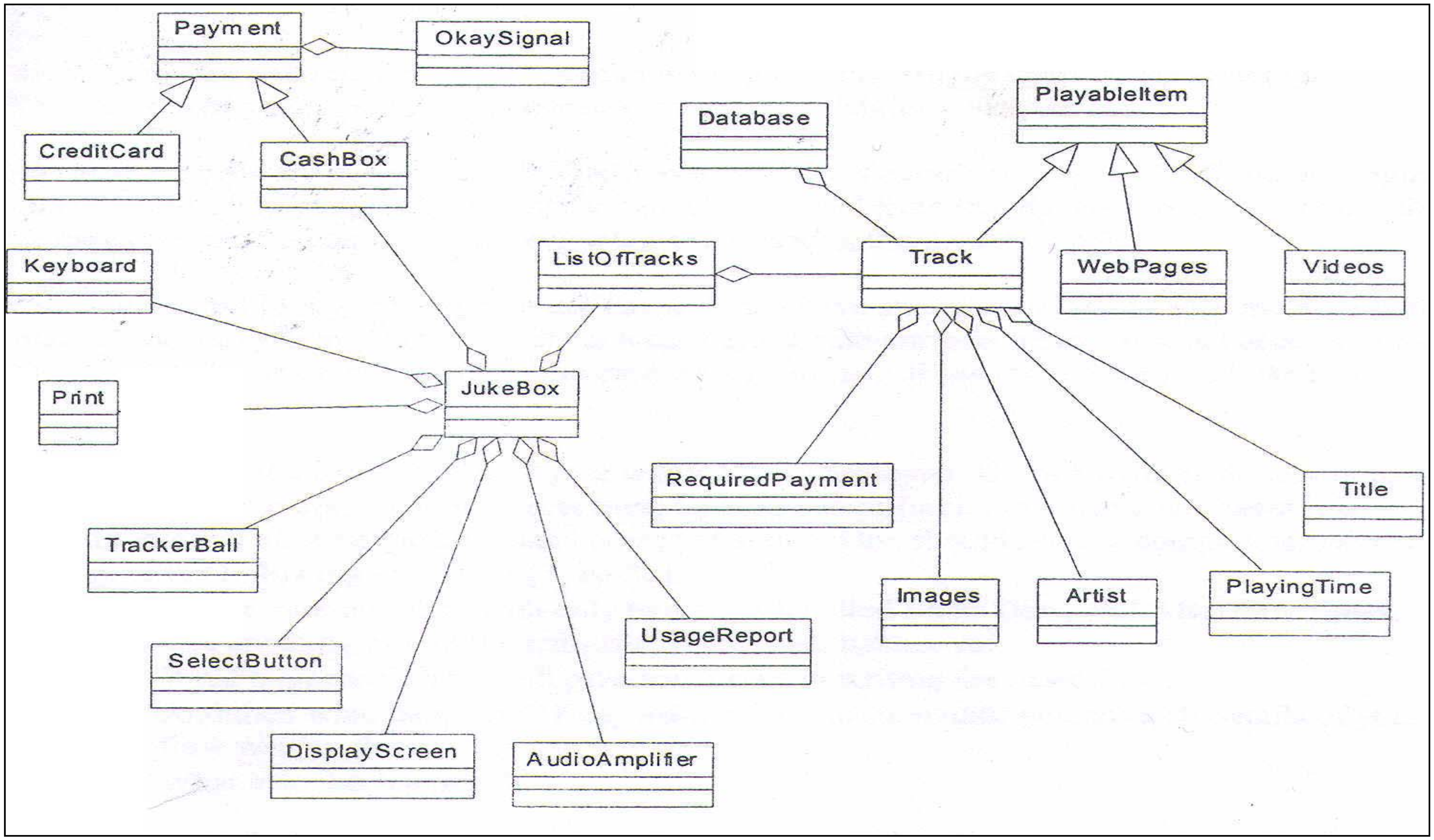


Figure 1.5 Domain models of the classes

Case Study: Use-case Modelling

The use-case analysis drives the entire process. The dynamic part of the UML model begins with the use-cases. The static structure is derived from the dynamic part. As we develop the use-case we should constantly be reviewing and updating the domain model.

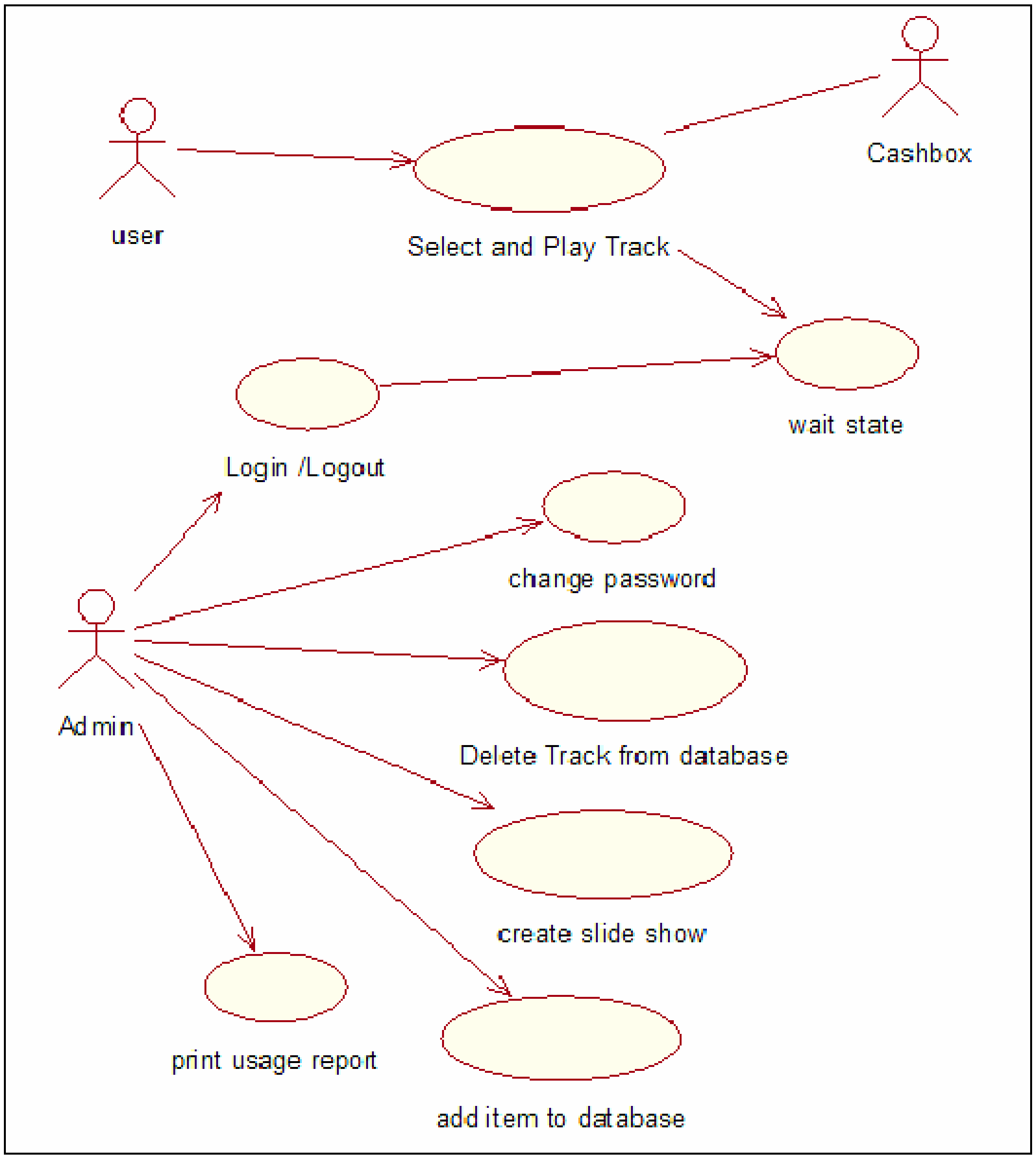


Figure 1.6 Use-Case Diagram

Once the use-cases are identified they are written out in basic text format. A basic template is used for each use-case:

- Basic flow - here you focus on what the user of the system is trying to do
- Alternate flow - In the alternate flow one considers the negative such aspects and anything else that happens other than the normal ideal sequence of events.

Consider the example where the text format for use-case Administrator Login is described:

Administrator Login

Basic flow: The Administrator attaches a keyboard to the system and strikes the "F1" key. The system asks the Administrator for his/her password. If this is valid the "Admin" interface appears on the display and the Administrator can proceed with other use-cases.

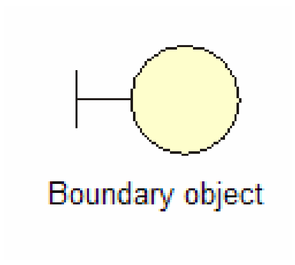
Alternate flows: If an invalid password is entered the Administrator is warned and is prompted again for the password.

Figure 1.7 Description of Use-case Administrator login

Case Study: Robustness Analysis

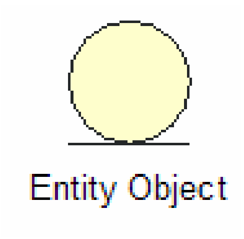
Robustness diagrams show the objects that participate in the scenario and how they interact with each other. The three different stereotypes used for representation are:

Boundary object:



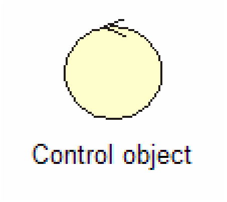
This is used by actors in communication with the system

Entity Object:



They are usually objects from the domain model.

Control Object:



These serve as a glue between the boundary objects and entity objects.

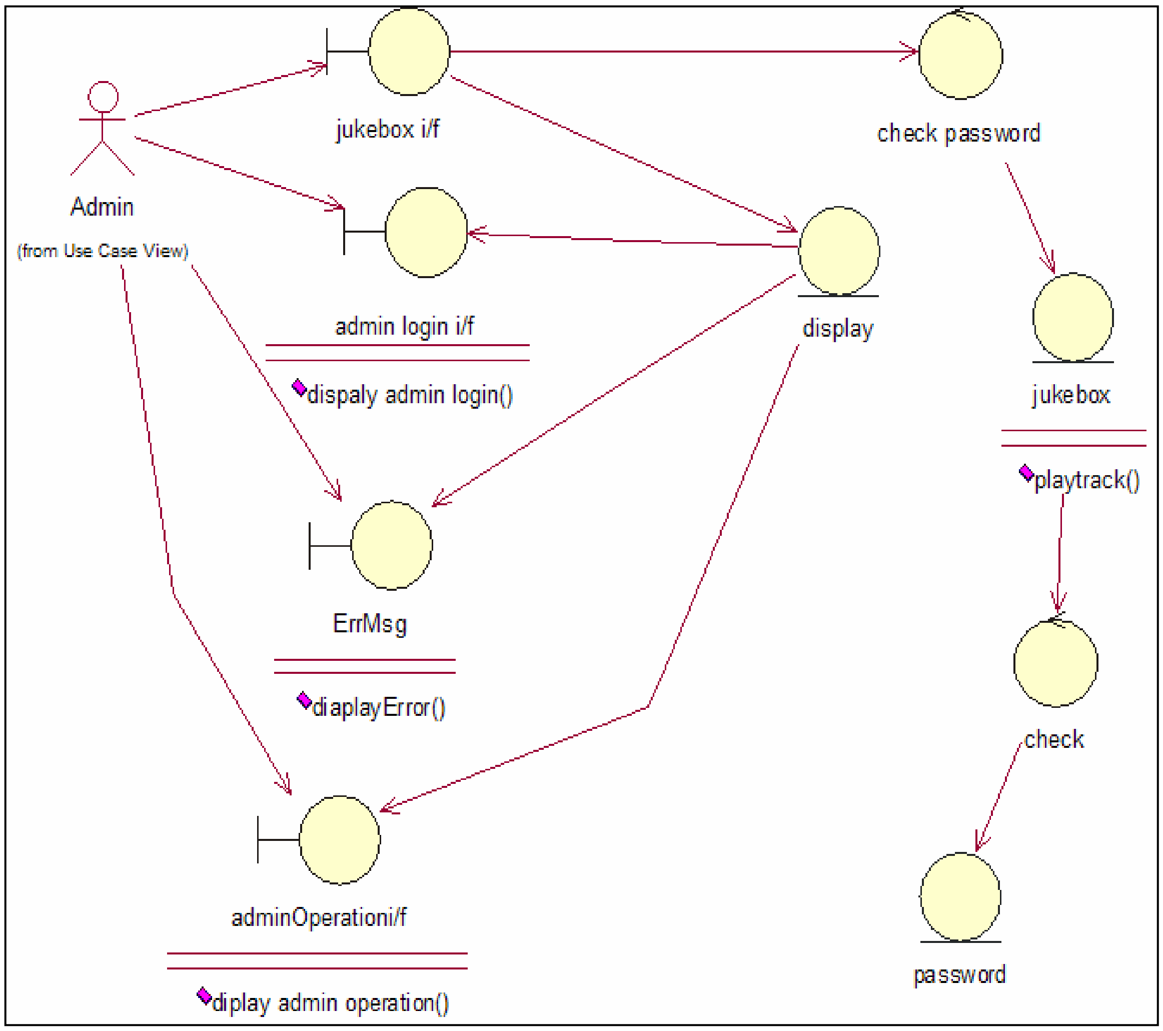


Figure 1.8 Robustness Diagram for Administrator Login

Case Study: Sequence Diagram

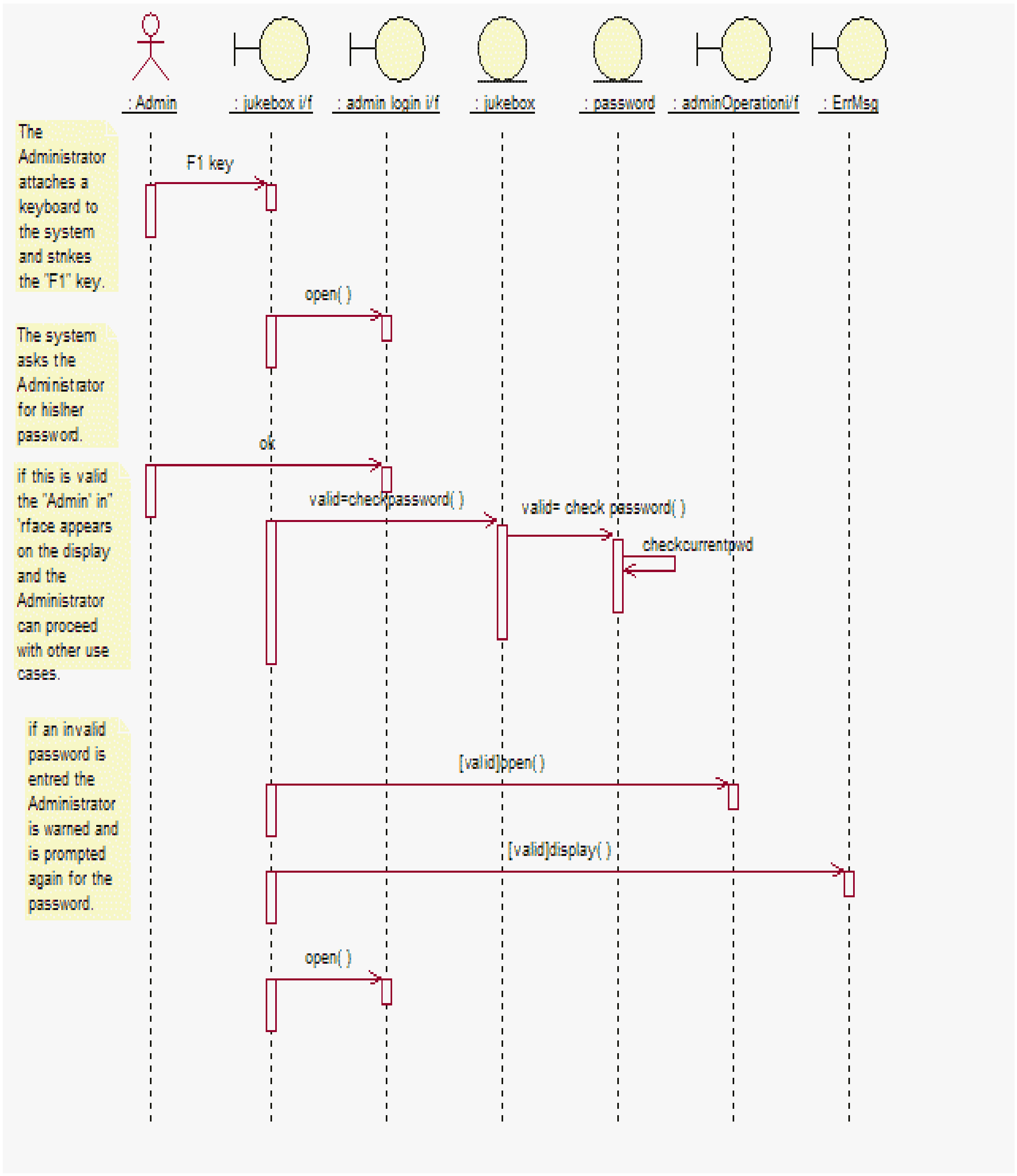

: Admin
: jukebox i/f
: admin login i/f
: jukebox
: password
: adminOperationi/f
: ErrMsg
The Administrator attaches a keyboard to the system and strikes the "F1" key.
F1 key
open( )
The system asks the Administrator for his/her password.
ok
valid=checkpassword( )
valid= check password( )
checkcurrentpwd
if this is valid the "Admin" in" 'rface appears on the display and the Administrator can proceed with other use cases.
if an invalid password is entred the Administrator is warned and is prompted again for the password.
[valid]open( )
[valid]display( )
open( )

Figure 1.9 Sequence diagram for the use-case Administrator login

Sequence diagrams are the first steps in detailed design. The focus is to allocate behaviours into the objects. The tree main goals of interaction modelling or sequence modelling are [12]:

- Allocate behaviour to boundary, entity and control objects

- Show the detailed actions that occur over time among the objects associated with each use-case.

- Finalize the distribution of operations among the classes. As you lay the detailed behaviour of the object in the sequence diagram you are finalizing the process of finding appropriate classes for both attributes and operations.

Case Study: Class Diagram

The final step is to draw the class diagram and this is the blue print of the system after critical design review is done on it. The Quality of the classes in the diagram is attained by ensuring the following features class diagram.

- Coupling
- Cohesion
- Sufficiency
- Completeness
- Primitivenes

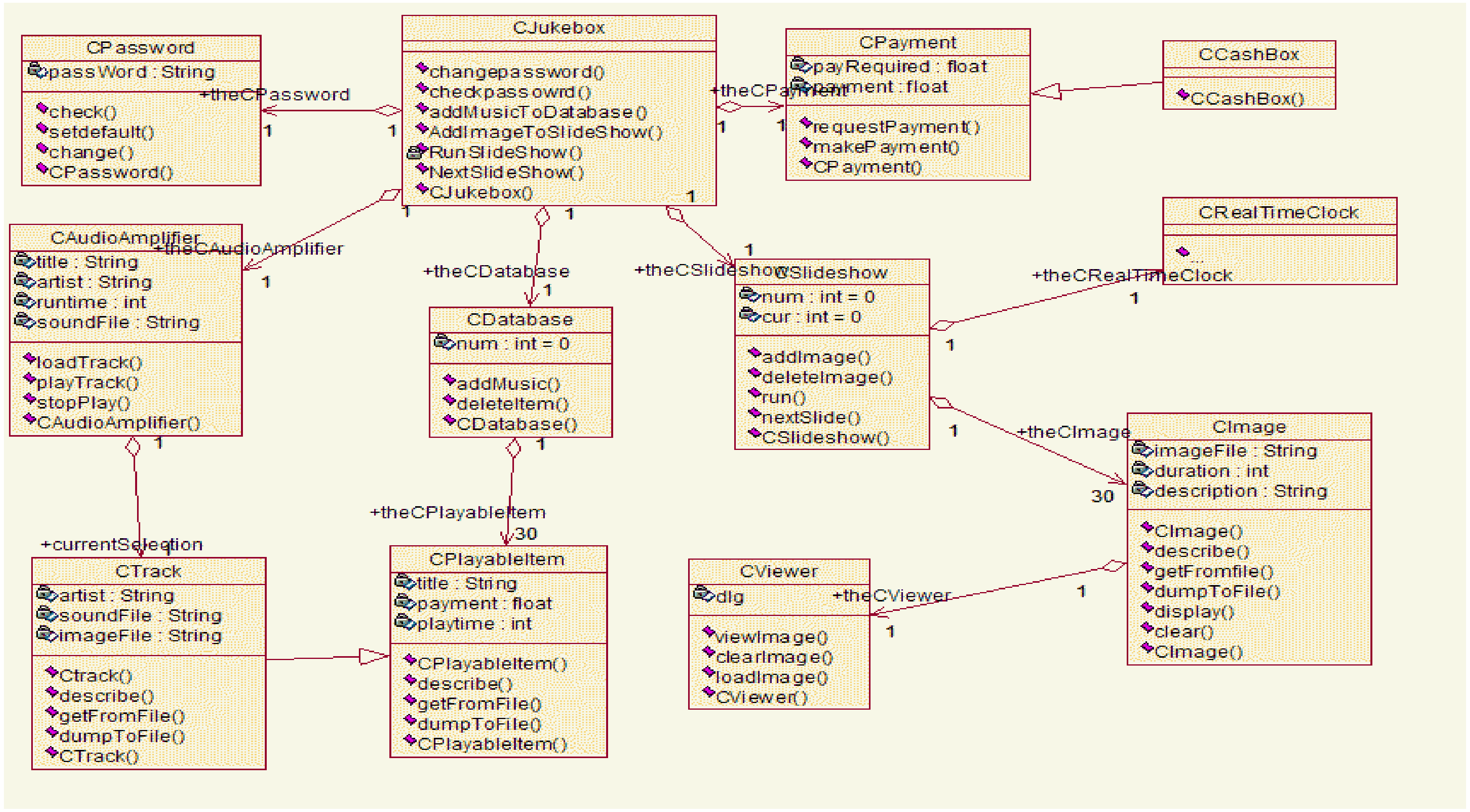


Figure 1.10 Final Class diagram of Jukebox [Adapted to JAVA from Ref[2]]

1.4 Conclusion

A fairly complex system of a jukebox was chosen as a case study. This was used for the design of an object oriented system and study of the UML tools. It gives a brief description of the system. Then using the case study the steps of ICONIX are covered. Finally from the ICONIX process we have a class diagram. This is the detailed design of the system and the blue print for which the implementation is done.

List of Figures

List of Tables

APPENDIX II

EVALUATION METHODOLOGY

Table of Contents

2.1 Introduction

There are many features that a UML tool can have but there are some which are necessary. There are a set of features that a CASE tool which claims to be a UML tool should ideally have. This section elaborates on a set of features that a tool should have and defines each feature. Further it also discusses a method to quantify the process of evaluating the tools. The rating system used for this evaluation is also discussed in detail.

2.2 Methodology For Evaluation

A standard application is chosen. This application is discussed in detail in Appendix 1. The application is complex and involves most of the concepts of an object-oriented system. Then the next step is to pick, one of the tools used in the evaluation, and develop the design for the system. This is done using UML as the modelling language and ICONIX is the process followed. Then the code generation is done using the tool selected. Once the code generation is done the logical part is coded in and then it is tested. This is repeated for all the four tools. Each feature is checked against each tool and compared, hence coming up with results and conclusions.

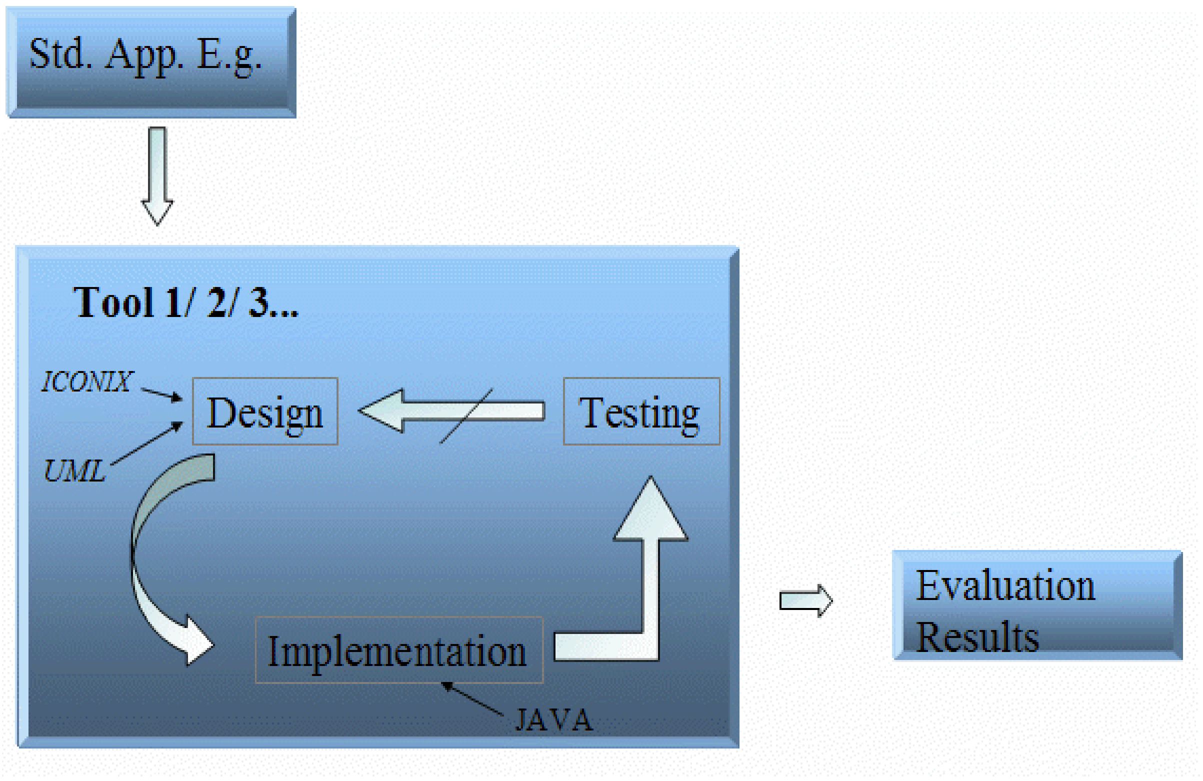


Figure 2.1 The method of evaluation

2.3 Evaluation Criteria

2.3.1 Repository

For a large project, a repository is necessary for the sharing of component designs between developers. Two or more developers can share components of a model or even collaborate on the development of a single component by defining ownership and sharing rights at the appropriate level. A repository is generally built on top of a database, which provides data sharing and concurrency control features. By providing locking and read-only access, the repository permits one developer to own the model while allowing others to read the model and its components, as well as to incorporate these components into their own designs.

Another way to build the repository is on top of the source code for a project, using a source-code control system to provide concurrency control. The benefit of this approach is a higher degree of synchronization between the code and the model. Another benefit is the elimination of yet another data source. Don't forget that if you use a database for a repository you must back-up this data store separately and perform three-way synchronization between the model, the repository and the source-code instead of just a two-way synchronization between the code and the model. With modelling tools that support a repository, changes to any component should be automatically propagated to any design which imports the component.

2.3.2 Customisation

It can be useful for a developer to be able to configure the tool to conform to some specific standards, perhaps company requirements or merely personal preferences, hence we would expect the tool to possess a certain level of customisability. This could include the options to view different panes, tools etc.

2.3.3 HTML Documentation

The object modelling tool should provide seamless generation of HTML documentation for an object model and its components. HTML documentation provides a static view of the object model that any developer using the model can refer to quickly in a browser, without having to launch the modelling tool itself. Also, by producing HTML as documentation, the number of required licenses for the modelling tool can be reduced by the number of people that need read-only access to the model information. The HTML documentation should include a bitmap picture of each of the diagrams in the model and should provide navigation throughout the model through the use of hyperlinks. The amount of time required to generate the HTML should be reasonable. A number of products available today support these features with varying amounts of success.

2.3.4 Usability

2.3.4.1 First Contact

The users initial experience with the system should be one in which the task of initially constructing and modelling parts of the system is easy. This feature is analyzed based on the user's first experience with the tool without referring to any documentation.

2.3.4.2 Easy to Use

The tool should remain easy to use even when dealing with complex diagrams and objects. For this criterion the focus is the tool's ability to hide and reveal information allowing the user to focus on specific details.

2.3.4.3 Pick List

The modelling tool should provide pick lists in several key interfaces: Collaboration and Sequence Diagrams - The tool should allow an object to be assigned to a class from a list of the classes in the model. It should allow the messages sent between objects to be chosen from a valid list of methods for the object (class) which is receiving the message. The pick list feature contributes significantly to the intuitiveness of the modelling tool and may be considered a must-have feature. The development of sequence and collaboration diagrams is greatly facilitated by being able to quickly select the message you want to send from one object to another.

2.3.4.4 Interface Presentation

The interface should have a layout which is both consistent and aesthetically pleasing. This category typically examines areas such as fonts, labels on diagrams, facilities provided for viewing diagrams, and the visibility of different states of the model.

2.3.4.5 Documentation and Help Files

Good documentation and search facilities help the user to learn the tool and feel comfortable around the tool. This determines how quickly a user can perform functions using the tool. Distinguishing the menus and the submenus makes readability easier. Name completion facilities, short-cut keys and learning aids can heavily influence the speed with which a user can perform common tasks.

2.3.5 Printing Support

The modelling tool should allow accurate renditions of large diagrams to be produced through multi-page printing. Print preview and scaling functionality should be supported to allow ease of fitting the diagram to the desired number of pages. The ability to fit a diagram to a single page is high on this list.

2.3.6 Exporting Diagrams

One key feature that is often overlooked is the ability to export diagrams into a format that may be imported into either a word processing document or a web page. The most popular graphics formats used for export are GIF, PNG and JPEG. When exporting, the tool should allow you to define the preferred resolution and size of the graphic that is produced. This functionality helps to include diagrams when writing reports, UML books or even display the design diagrams on a web site.

The XMI standard from the Object Management Group (OMG) is one of the recent developments in the UML developer community. XMI is an interchange format which has the potential to finally allow seamless sharing of models between best-of-breed development tools. For example, rather than writing scripts within a UML modelling tool to create reports, instead a user could simply export the model under development using XMI and import the model into a specialized report writing tool. It is therefore good that a tool should be able to export diagrams to XMI format.

2.3.7 Robustness

A UML tool should have rock-solid reliability and consistency. This is to prevent users from losing potentially hours of productivity, when the tool crashes in the middle of a design session, or corrupts a model which hasn't been backed up. A tool which causes hours of work to be lost due to a crash or file corruption is very unsatisfactory. As a developer, you know the feeling of disdain for 'productivity applications' that are less productive than raw coding tools. If you are a manager, you have seen the resentment developers will show when being required to use an unreliable tool.

Another strategy to apply here, which is recommended that tool vendors adopt, is borrowed from office productivity applications. The strategy is to have the UML tool to automatically save a model in the background at periodic intervals.

2.3.8 New Release

The modelling tool selected should continue to be actively improved through bug fixes, performance improvements, and the addition of new features. After all, you are making a big investment in time and money and it is not easy to change to another modelling tool. Another factor is that these tools have to adapt to the new technological advancements in hardware and software environments.

We can determine if a product is evolving by enquiring for a detailed schedule of recent releases and a roadmap for the product's future and by looking closely at the rate at which features and improvements have been made. One may also look on the company website for the product announcements and outside reviews.

New versions and improvements in functionalities are good, provided the new version of the tool is backward compatible with the older versions. It is unreasonable that one should be stranded with old design diagrams which are not compatible with new versions of the tool. So it is necessary to consider this feature when deciding on a tool.

2.3.9 Round Trip Engineering

The ability to both forward and reverse engineer source code (Java, C++, and CORBA IDL) is a complex requirement that vendors support with varying degrees of success. The successful combination of these two features, forward and reverse engineering is defined as round-trip engineering. [1]

2.3.9.1 Code Generation or Forward Engineering

Code Generation is the process of generating code in the respective programming language for the classes, attributes and operations defined in the design. It is also possible to generate the code for the relationship between the classes and other components. Once this is generated the programmer just has to implement the logic of each operation. Forward engineering is very useful the first time that code is generated from a model. This will save you much of the mundane work of keying in classes, attributes and methods.

2.3.9.2 Reverse Engineering

Reverse engineering is the ability of the tool to recognise the new classes, methods and attributes that the programmer adds to the application. This can be done during the development of the application or during maintenance once the application is deployed. Reverse engineering is very useful both to transform code into a model when no model previously existed, as well as to resynchronize a model with the code at the end of an iteration.

During an iterative development cycle, once a model has been updated as part of the iteration, another round of forward engineering should allow code to be refreshed with any new classes, methods or attributes that have been added to the model. This step is less commonly adopted by developers because many tools can hopelessly mangle source code in the process. The problem is that the source code contains much more than the model; tools must be very adept at reconstructing the source code that existed prior to the new round of forward engineering.

At minimum, the modelling tool should successfully support forward engineering the first time and reverse engineering throughout the process. Also, the tool should have no trouble reverse engineering the full Java language. The way to verify this feature is to implement your own source code and try the round trip engineering on your code.

2.3.10 Data Modelling

The object modelling tool should allow integration with data modelling facilities. There are many ways to provide this functionality. One way is for the UML tool to provide a feature allowing an object model to be transformed into DDL, which is the SQL needed to create tables for classes. Another way is for the UML tool to export metadata to a data modelling tool which can import the metadata and use it as the basis for a data model. An advanced, integrated set of tools should allow the data models and object models to be synchronized after each iteration of the design.

2.3.11 Model navigation

The modelling tool should provide strong navigational support to allow a developer to navigate through all the diagrams and classes in the model. A directory or pick list of classes sorted by name is one way to allow a designer to jump to the desired class on a diagram.

For large diagrams, the tool should provide ease of navigation when zooming and panning. The tool should also allow ease of navigating to the source code for a class when round-trip engineering is being used.

2.3.12 Diagram views

The modelling tool should facilitate customization of the view of a class and its details. For instance, it should be possible to exclude all get/set methods from the diagram since they tend to clutter, rather than clarify a diagram. The full signature of methods should be allowed to be shown or hidden easily, depending on the level of detail desired. The visibility of attributes and methods (private, protected, public) should be another dimension used to select what to show or hide on the diagram.

2.3.13 Platform

In order to maximize an investment in a modelling tool, one has to carefully consider the platforms on which the tool will run. Java's Swing user interface allows cross-platform graphical user interface (GUI). So if the UML tool is built using a cross platform GUI, one can look over the issue of platforms.

However, cross-platform tools need to be supported on commodity platforms such as Linux in order to achieve large-scale adoption by programmers. Sun had originally done little to promote Java on Linux. But recent industry initiatives, principally from IBM, which has pledged broad-based support for Linux on all of its hardware platforms and is supporting the Apache/Jakarta project, are now rapidly pushing Java onto Linux. Perhaps because IBM has moved to distribute its version of JDK 1.1.8 to the major Linux vendors, Sun has been compelled to support the distribution of a fully functional JDK 1.2 (Java 2, with Swing) for Linux. This Java port to Linux has been largely accomplished through the efforts of the Blackdown Group. So a tool developed on a platform independent programming environment, gives a user the freedom to adopt any platform.

2.3.14 Multi-user Support

When working in a team oriented environment it is essential that the tool provides support for multiple users. This support is generally required in the form of multiple user access to the development software which in turn requires users to be constrained by predefined permissions. The changes made by each user to a model should be backed up and made available to the next user.

2.3.15 UML Support

While many tools claim full support for UML 1.3, in reality this is a complex requirement and some tools may not live up to advertised claims for full support. At minimum, the diagrams which should be supported are the Use-case, Class, Collaboration, Sequence, Package, and State diagrams.

2.3.16 Support for Language

This feature deals with the different languages supported for code generation. Some of the common languages that are supported by tools include: JAVA, C++, VC++, COBRA, ADA, J2EE, C #, Visual Basic.net, CORBA IDL and Visual Basic 6

2.3.17 Installation

Installation of any tool should be fairly easy and the tool should be up and running without many hassles. One should be aware of all the prerequisites before considering a tool. Some tools require a database for it as a repository while others require an application server. It is essential to study the kind of application you develop before selecting a tool. Most of the prerequisites are open source software.

2.3.18 Class Diagram Features

- **Class box size flexibility :**
- **Line flexibility :**
- **Independent placement of association end names:**
- **Preservation of position of end names and multiplicity labels:**
- **Moderate binding of relationship lines:**
- **Distinguish between remove from diagram and delete from model:**
- **Restoration of relationships in new diagrams:**
- **N-ary associations:** An N-ary association is an association among three or more classifiers (a single classifier may appear more than once). Each instance of the association is an n-tuple of values from the respective classifier. A binary a
- Association is a special case with its own notation.
- **Undo functionality for diagrams**
- **UML profiles (stereotypes and tagged values)**: A UML profile is made up of one or more “stereotypes” that may have “tagged values” and “constraints” [24]. Profiles are sometimes referred to as the

'lightweight' built-in extension mechanisms of UML, in contrast with the 'heavyweight' extensibility mechanism as defined by the MOF specification. This is because there are restrictions on how UML profiles can extend the UML meta-model [25]. These restrictions are intended to ensure that any extensions defined by a UML profile are purely additive [6].

2.3.19 Support Robustness Diagrams

In this research project the ICONIX methodology is used to do the design. Some tools do not support the different notations in the robustness analysis. This may not be an essential feature but for companies that use these methodologies as one of their standards one should take this feature into consideration.

When considering this feature one should analyse the following: the tools support the notations used to represent the types of stereotypes, the rules and constraints applied while drawing the diagrams.

2.4 Evaluation process

The set of features was drawn up and defined. Each feature is defined clearly so as to eradicate duplication of features and definition of these features. It also clearly sets a boundary around each feature. Each feature can be further divided into sub features. For example consider the feature round trip engineering. This can be subdivided as shown in Table2.1

Features	Sub-Features
ROUND TRIP ENGINEERING	• Code Generation • Reverse Engineering • Synchronization With Editor

Table 2.1 Feature divided to subfeatures

The evaluation is done using a “rating system” where a weighting is given for each feature and a rating is given based on the tool. This kind of evaluation gives a weighting for each feature based on its significance. This gives a user the flexibility to change the weightings and hence evaluate the tools. Another user can prioritise the features based on the technical and functional requirements of the product developed.

2.4.1 Rating Method

The weighting for each sub-feature is fixed based on the significance and the importance of each of the feature when compared with the rest of the sub-features. This prevents some tools to score a higher value for a feature which is not essential. If a feature is absolutely essential then it is assigned a three. If the feature pleases the eye yet it is not significant in terms of production then it is given a 1. The weightings given to the features in this research are based on what I felt was significant. This judgement was based on extensive literature survey, complaint from forums and discussion with users. The weightings are shown in Table 2.2

Nice to have Feature	Good Feature	Essential Feature
1	2	3

Table 2.2Weightings for feature

The ratings for the features are given based on how well the functionality is implemented, for each tool chosen. If a feature is poorly implemented then it is assigned 1 but if it is implemented well then it gets a 3 E.g. Consider the feature “The class box flexibility”. In some tools this feature is there yet it does not serve the purpose where, it should reveal all the attributes and methods on increasing the size. This is a poor implementation and it is assigned a one. Figure 2.2 shows the score for the ratings.

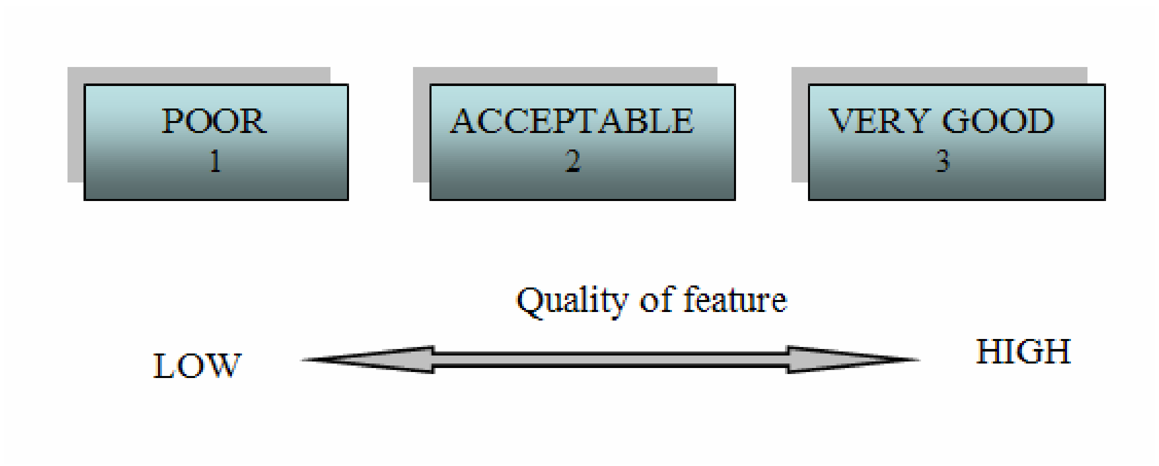


Figure 2.2 Score for ratings

The product of the weighting and rating is calculated for each feature and the total sum of the products gives the final score for each tool. This can be summarised into the following general formula:

Score for a tool = $\sum$ (weighting of each feature × rating of tool for that feature)

That is if weighting is represented in W and rating represented in R then the score of a tool 1 (T1) is obtained by

$$T1 = (W11 * R11) + (W12 * R12) + (W13 * R13) + \ldots\ldots$$

2.4.2 Evaluation Table

2.4.2.1 Features Evaluated using Implementation

The interface of the UML tool should be intuitive and easy to learn, in addition to providing required functionality. These aspects of a tool can be further subdivided into sub-categories for evaluation. The evaluation table is illustrated below (Table 2.3).

Main Features	Sub Features		Weightage
Repository	***Database***		*2*
Customisation	***Components and tools***		*2*
HTML Documentation	***Generate Web Reports***		*3*
	Save diagrams to include in reports		*3*
Usability	***First Contact***		*2*
	Ease of use		*3*
	Pick List	*Pick list of the classes in the model while drawing.*	*2*
		Tree structure in a pane where all the classes can be viewed.	*3*
		Select methods to draw sequence or collaboration diagram.	*3*
		Drag and drop	*3*
	Interface Presentation	*Select methods to draw sequence or collaboration diagram.*	*3*
		States are visibly distinct	*2*
	Documentation and Help Files	*step by step documentation*	*3*
		Online help	*2*
		Keyword search facilities	*3*
		Short – cut keys	*1*
Printing Support	***Fit diagram to a single page***		*3*
	Print preview		*3*
	Scaling functionality		*3*
Exporting Diagrams	***Save diagrams any picture editor format***		*3*
	XMI format		*2*
Robustness	***Does tool crash or corrupt diagrams***		*3*
	Automatic saving at periodic intervals		*1*
New Releases	***Any New version releases announced***		*3*
Round Trip Engineering	***Code generation***		*3*
	Reverse engineering		*3*
	Synchronisation with editor		*2*
Model navigation	***Zooming and panning***		*3*
	Navigating between source code and diagram		*2*

Diagram views	***Customizing details of classes***		***3***
	Reveal and hide methods		***3***
	Visibility of methods and attributes		***3***
UML Support	**Key Notational parts**	*Use case diagrams*	*3*
		Class diagrams	*3*
		Sequence diagrams	*3*
		Collaboration diagrams	*3*
		State diagrams	*3*
		Activity diagrams	*3*
		Component diagrams	*3*
		Deployment diagrams	*3*
		Package diagrams	*3*
	Class Diagram Functionalities	*Class box size flexibility*	*3*
		Line flexibility	*2*
		Independent placement of association end names.	*3*
		Independent placement of multiplicity labels.	*3*
		Preservation of position of end names and multiplicity labels.	*3*
		Distinguish between remove from diagram and delete from model.	*3*
		Restoration of relationships in new diagrams.	*3*
		N-ary associations	*3*
		Undo functionality for diagrams.	*2*
		UML profiles (sterotypes and tagged values)	*2*

Table 2.3 Features and Sub-Features Used For Evaluation

2.4.2.2 Features Evaluated without Implementation

The stages in the design and implementation of the jukebox example, was used to evaluate a chosen tool against most of the ideal features of a UML tool. Some features could not be evaluated based on the example because it was not within the scope and time limit of this research. Others like platform support and installation are not included in Table 2.3 since it is not a quantifiable feature. Hence it cannot be rated. These features are shown below:

- Installation
- Platform
- Support For Languages
- Data Modelling
- Multi-User support

2.5 Conclusion

There are many UML tools in the market, which claim to have certain features. Most of the time we find that these tools do not have the required features to satisfy our business needs while in other cases the vendors claim on the features might not be up completely up to the mark. This section has tried to come up with a set of features that an ideal UML tool should have. It also gives a rating method whereby the user can get a quantified result on the tools he evaluates. In the rating method one can prioritize the features according to the business claims and assess each tool.

The evaluation tool or matrix [Appendix VI] emphasizes on the technical features in a UML tool. These technical features are mainly functional and usability issues. Non-technical aspects such as cost, training, local support, availability of resource and developmental environment, are not part of the comparison matrix because it is beyond the scope of this project report.

This research report is unique by presenting an evaluation tool to compare any set of UML tool. The tool has the following headers: Features, Weightings, UML Tools used for comparisons. The Features listed are indispensable in any UML tool. The weightings given for each feature could be customised according to the user's priority list. And finally a set of UML tools can be used for comparison. This can be done by plugging in quantitative values and calculating the highest score.

List of Figures

List of Tables

APPENDIX III

UML TOOLS USED FOR EVALUATION

Table of Contents

3.1 Introduction

This section talks about the tools that were used for evaluation in this research work. The tools that are discussed here include ArgoUML, MasterCraft, Rational Rose and Together Control Center. These tools are considered as mainstream Object Oriented tools and they also support implementation in JAVA language. UML tools can be classified as follows:

- Basic diagram-drawing tools
    - e.g. Visio (basic version).
- Main-stream OO CASE tools
    - e.g. Together, ArgoUML.
- Specialist real-time/embedded tools e.g. Rhapsody, Telelogic. [13]

3.2 ArgoUML v 0.14

ArgoULM is an open source project developed by Jason Robbins and David Redmiles at the University of California. According to Greek mythology, the hero Jason built a ship called Argo and with his comrades, the Argonauts, he left for the quest of the Golden Fleece [16].That is how they came up with the name for this tool. ArgoUML is completely implemented in JAVA. Since it is byte code-interpreted the speed of execution is not electrifying. [18]

ArgoUML was conceived as a tool and environment for use in the analysis and design of object-oriented software systems. In this sense it is similar to many of the commercial CASE tools that are sold as tools for modelling software systems. ArgoUML has a number of very important distinctions from many of these tools. These include the following:

- ArgoUML draws on research in cognitive psychology to provide novel features that increase productivity by supporting the cognitive needs of object-oriented software designers and architects.
- ArgoUML supports open standards extensively—UML, XMI, SVG, OCL and others. In this respect, ArgoUML is still ahead of many commercial tools.
- ArgoUML is a pure Java application. This allows ArgoUML to run on all platforms for which a reliable port of the Java2 platform is available.
- ArgoUML is an open source project. The availability of the source ensures that a new generation of software designers and researchers now have a proven framework from which they can drive the development and evolution of CASE tool technologies.

ArgoUML meets the OMG standard for UML 1.3 and supports as diagram types class, state machine, use-case, collaboration, activity and object/ component/ deployment diagrams. It is only sequence type diagrams that are not supported in ArgoUML. For code generation, ArgoUML supports only Java and reverse engineering is not supported well in ArgoUML. The documentation and user manuals for ArgoUML are not complete.

3.3 Together Control Center V6.1

Together Control Center includes the features you need to build enterprise level applications, allowing the entire development team to collaborate using common language, diagrams, and software. This product is

developed and maintained by Borland. Like ArgoUML, Together is also byte code-interpreted. Hence there is a sluggish rate of execution on ordinary hardware. It requires JDK 1.3 as a virtual Java machine.

Modelling support includes all the standard UML diagrams, plus additional diagrams for other special types of modelling. This includes support for class diagrams and UML 1.4 diagram types like use-case, sequence, collaboration, state, activity, component and deployment for modelling. Class and sequence diagrams generate source code automatically and keep it in synchronization. Code generation can be done in Java and C++, and reverse and roundtrip engineering as well as team support are provided.

Together Control Center provides an efficient feature called simultaneous round-trip technology. Unique simultaneous round-trip technology means that changes to application code are immediately reflected in visual models [18].Make changes to the model or the source code and each stays synchronized with each other. Plus, Live Source provides visibility into existing applications, generating class models instantly. Together Control Center, applications can be built for one application server and easily switched to another, protecting development assets even if server changes are required.

Another option is that of direct import of existing relations from a database as ER diagrams. Via a dialog window, the necessary settings (server type, database name, host, port, username and password) can be made for database communication. Databases supported are Oracle 7.3.x/8.x, DB2, MySQL, MS SQL, Cloudscape, ODBC/Access 97 and SequeLink/Oracle.

Usability issues on picking a class from a list have to be modified further. Multi-user support though available it is not very efficient. Since UML is most efficient in large systems and there are multiple developers working on these systems one cannot ignore the efficiency of the multi-user support provided by the tool.

3.4 RATIONAL ROSE ENTERPRISE EDITION V. 2001A.04.00

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 Modelling Language (UML), Component Object Modelling (COM), Object Modelling Technique (OMT), and Booch '93 method for visual modelling [4].Using semantic information it ensures correctness by construction and maintaining consistency.

Of the CASE tools in the test field, Rose supports most languages: Java, C++, ADA 83, ADA 95 and CORBA IDL and DDL for database applications. Rose offers both roundtrip and reverse engineering. Rational Rose provides the following main features to facilitate the analysis, design, and iterative construction of your applications:

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

Rose claims to support multi-users and developer groups. Rose makes a private working   for all developers, in which each has an individual via of the whole model. Modifications are thus restricted to the private working area until they are checked in to the CMVC (Configuration Management and Version Control System). Yet there are many issues around the efficiency Rational Rose provides for multi-user and reverse engineering. (See [20])

3.5 MASTERCRAFT FOR JAVA V 6.0

MasterCraft Enterprise Java is an integrated tool suite developed to increase the productivity and quality of large, multiple teams working on complex, mission critical application development. It provides an object oriented environment that supports a UML based Component Modeller and Repository-driven development process [27].

MasterCraft's modelling tools, in MDA™ like approach, allow the designer to keep the logical application independent of the underlying technology. Platform-specific generators deliver code for the required platform, using the models. MasterCraft supports a component-based repository driven development process and has a visual modelling tool, a GUI modeller and an object-oriented specification language.

MasterCraft supports development of applications, which use geographically distributed resources and allow geographically distributed end users. MasterCraft uses a central repository, to manage all information. It also integrates with and processes information stored outside the repository database. The meta-model integrates meta-data across the various phases of the life cycle.

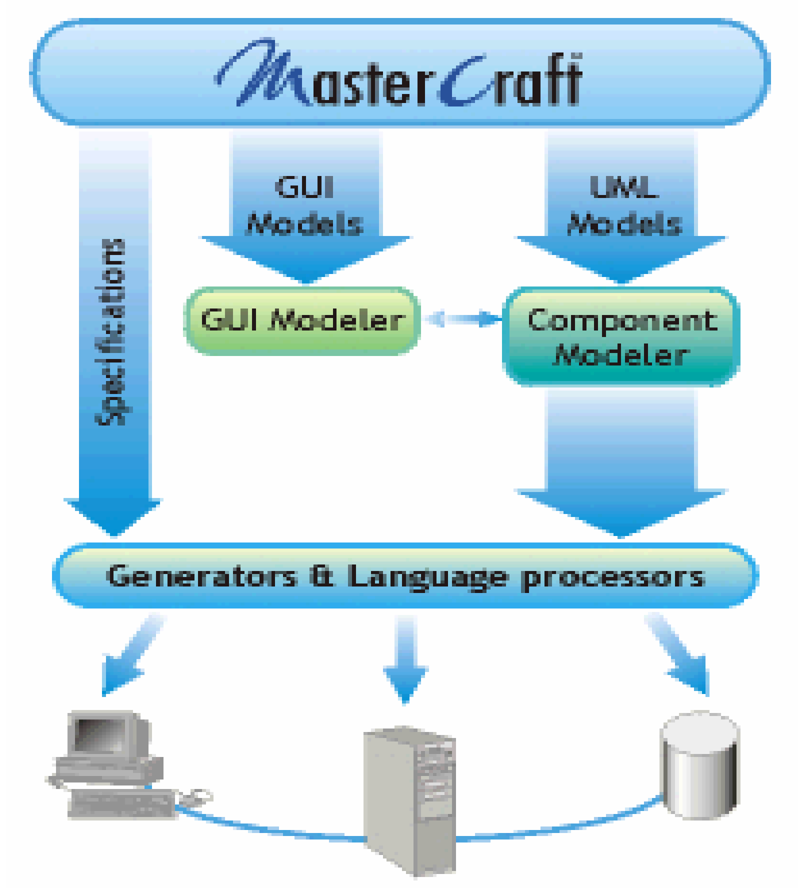


Figure 3.1 MasterCraft – Integrated Tool-suite

The Component Modeller in MasterCraft is the visual modelling and repository tool; MasterCraft uses the repository to create, validate, and store the analysis and design models. MasterCraft's GUI Modeller is used to model, validate, and generate the Graphical User Interfaces of the application. (See figure 3.1)

MasterCraft supports the development process through a set of pre-defined roles. Software builds, releases and versions are managed through clear processes. These processes are configurable and can be integrated into the organisation's environment. Roles in MasterCraft are sets of logically coherent tasks that developers perform during the software development life cycle (SDLC).

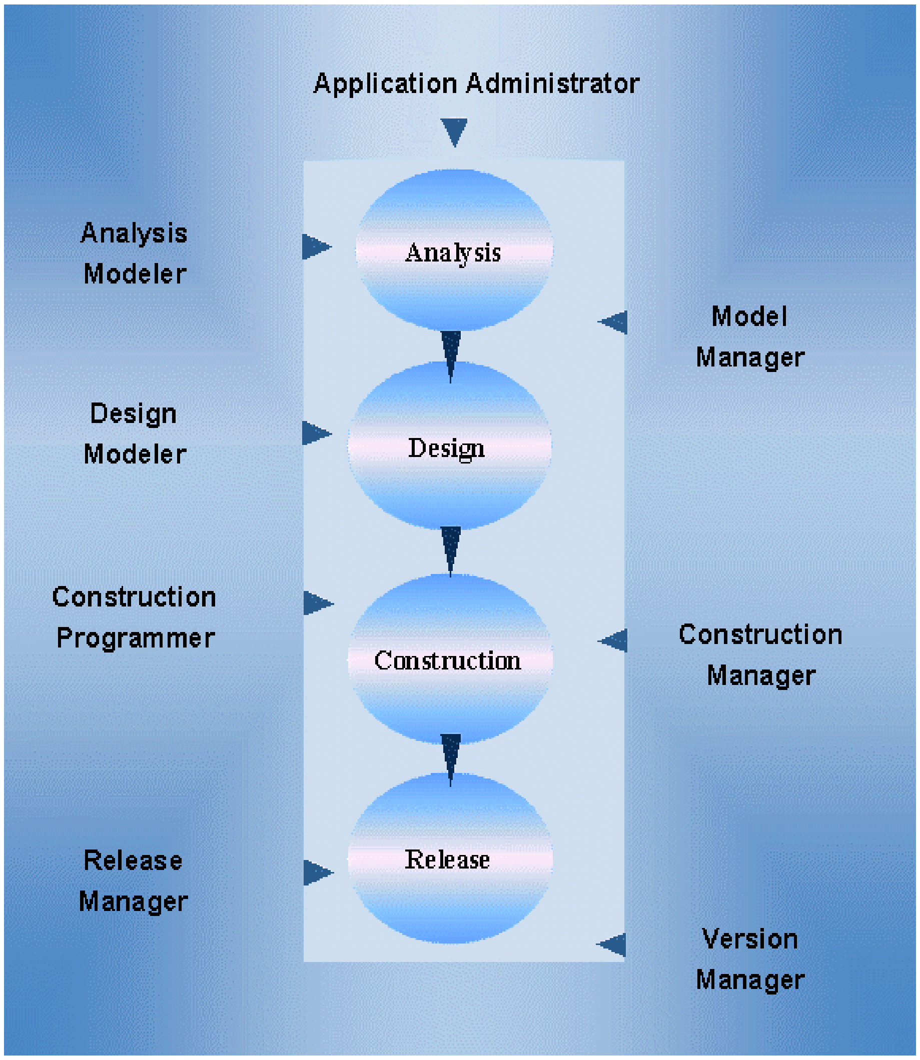


Figure 3.2 Role-based Development

MasterCraft supports the entire Software Development Life Cycle through the Analysis, Design, Construction, Release and Maintenance phases. One or more pre-defined roles map to one or more phases of the life cycle.

Mastercraft allows hiding the logical part from the technology. This is done using Q++. Q++ has the following features:

- Reduce Coding
- Hide Technology: there are interpreters that convert the Q++ code to the underlying technology like C++, JAVA etc..
- Stronger Type Checking
- Maximum use of modelled information in repository

Master craft allows unit testing up to the level of each operation. It allows reverse engineering but this is not reflected back into the diagram automatically. All the new classifiers added are saved in the repository and the information can be browsed. But it has to be updated manually in the diagram in this version of Mastercraft. The installation and setup of this tool requires support and cannot be easily done the first time.

3.6 Conclusion

There are many tools in the market available for a user to pick from but one has to spend time to study the kind of tool you need for your business requirements. Each tool has positive and negative aspects to it. The section discusses the four UML tools used for evaluation in this research work. For each tool the additional features and weaknesses for the tools are discussed. Further details on the tools can be obtained from the respective websites for each of these tools.

List of Figures

APPENDIX IV

UML TOOLS – EVALUATION OF MODELLING

Table of Contents

4.1 Introduction

UML tools are mainly modelling tools and hence modelling is an inevitable issue for discussion. This section reviews the interfaces for each of the tool describing flexibility, navigability and architectural issues. It further discusses UML notations like class diagrams, and sequence diagrams in the different tools.

4.2 General View of each tool

The usability and layout of a tool is critical to a user. A tool might provide all the necessary features but if it is not user-friendly, a user will be very hesitant to use the tool. The usability of the four different tools is discussed in this section. It also describes the flexibility allowed by the tools in the toolbar, the different panes etc. The snapshot of the respective interfaces of each of the tools is shown. Aspects such as navigation are also applicable when covering usability.

4.3 Together

4.3.1 Modular architecture

Together consists of a large set of modules representing available features. A given project will not require all the available features. Some feature modules can be turned on or off as necessary. Therefore unneeded features are not loaded, simplifying the user interface by displaying only those commands and features you need for your project. Features are activated on several levels:

- Your selected *user role* determines which features are available for all projects.
- The *Activate/Deactivate* Features dialog enables you to set which features to load for the current project.
- Features activated on *demand* are loaded automatically when they are needed. [43]

Each feature module has a configuration file that stores property settings that apply to the feature. You can use the Options dialog to make configuration settings.

4.3.2 Flexibility : User Interface

The Together user interface (including menus, toolbars, and panes) changes according to how you are working with Together.
The menus, toolbars, and panes available depend on several factors:

- your user role
- the project context (whether a project is open, and which feature modules are activated)
- the selected workspace in the project. [43]

Roles
During installation, you choose a user role (Business Modeler, Designer, Developer, or Programmer). When Together starts up it shows only those menu commands, toolbars, and panes that are appropriate for your chosen role. The role also determines the default workspace. The user role is a global configuration setting that you can reset in the Options dialog.

Workspaces

A workspace is an arrangement of panes that you can save and reuse as you find convenient. Workspaces are saved with the project.

4.3.3 Navigation: Together Main Window

The ease of navigating through the diagram and the element properties is a critical issue. This widely determines if the tool is user-friendly or not. Figure4.1 gives us an outlook of the Together front-end. The main window is divided into four major panes.

- Explorer: for file system and project navigation.
- Designer: for creating UML and other kinds of model diagrams as well as for building graphical user interfaces. The Designer has a toolbox for its GUI construction tools.

- Editor: for viewing and editing source code files and other text files.
- Message pane: for system messages, special tasks, and results of some feature operations.

The *focus pane*, with the light blue title bar, is the site of the most recent activity. Together elements are individual components of the project and the user interface. Elements can be diagrams, files, diagram elements such as nodes or links, names, error messages, and so on. Right clicking on a Together element displays a menu of commands for that element. These right-click menus vary according to the type of element. Many elements have Inspectors for accessing the elements' properties. You can display the Inspector of an element by selecting Properties from its right-click menu. Many elements have Property Inspectors for accessing the elements' properties. Property inspectors enable you to view and change the properties of many Together elements. Inspectors are organized into tabbed pages whose content depends on the type of element. [43, 45]

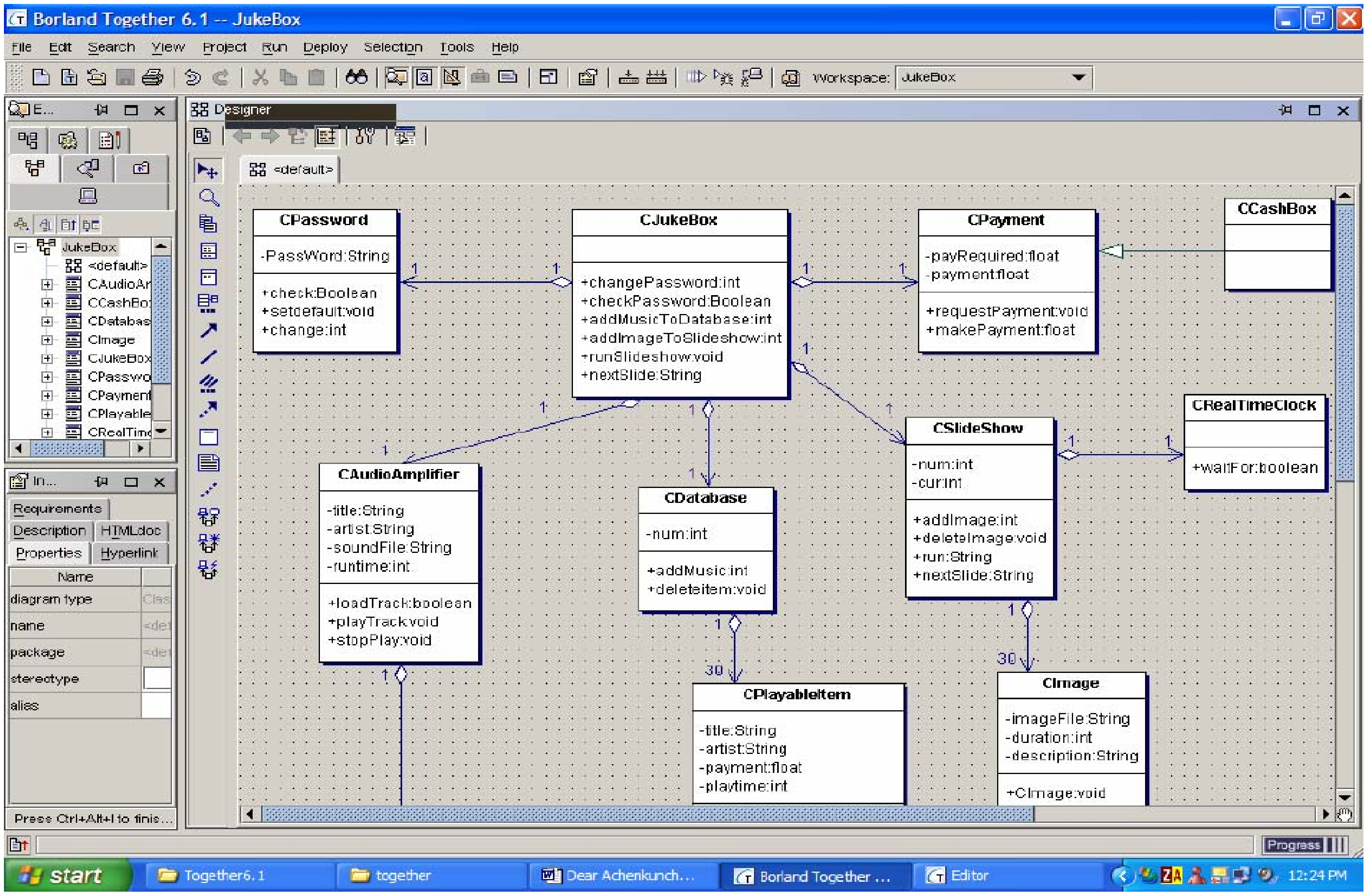


Figure 4.1Workspace in Together Control Center

4.4 Rational

4.4.1 Navigation :Rational Rose Main Window

Rational Rose’s graphical user interface is used 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 [19]

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

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. 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 behaviour of the element.

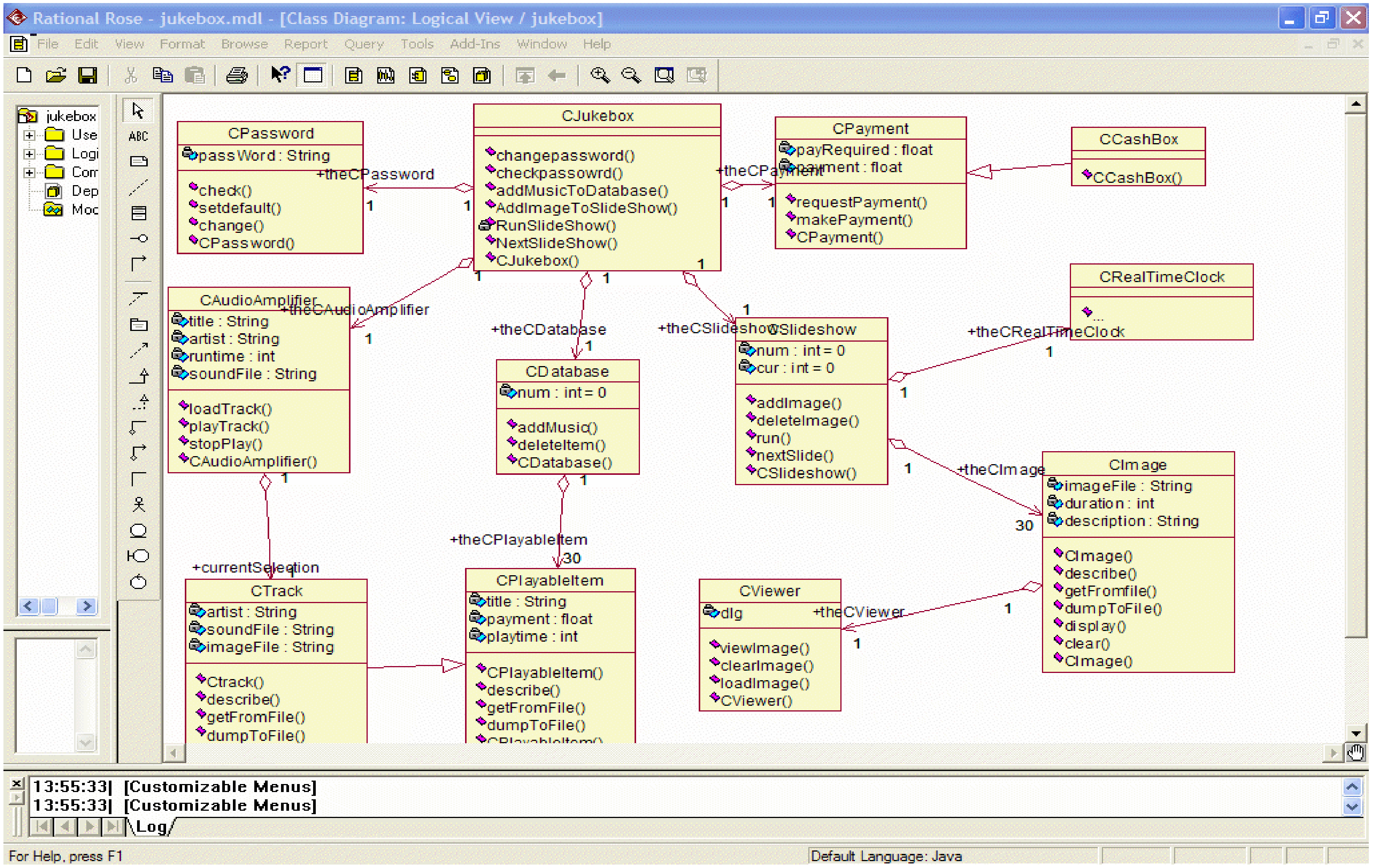
The documentation window is used to describe model elements or relationships. The description can include such information as the roles, keys, constraints, purpose, and essential behaviour of the element. Rose uses the log window to report progress, results, and errors that occur as a result of a command or action in your model. The messages posted to the log are prefixed with a time stamp. This enables you to keep track of when an event or action occurred. Like the documentation window, the log window can be docked or floating.

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.

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.

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.

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.



```
13:55:33|  [Customizable Menus]
13:55:33|  [Customizable Menus]
```

Figure4.2 Workspace in Rational Rose

4.5 ArgoUML

4.5.1 Navigation : ArgoUML

The user workspace of argoUML is shown in figure 4.3 below. At the top of screen is a menu bar. Under that there are Toolbars. Then the bulk of the window comprises four subwindows or Panes. Clockwise from top left these are:

- Explore pane
- Editing Pane
- Details Pane
- To-Do Pane

At the top of the Editing Pane is another toolbar called the Edit Pane Toolbar. Finally at the bottom of the window is a status bar. [17]

Explorer allows you to navigate through our model. This pane lists all the classes, interfaces and data types of our model as a tree view.
The Editing Pane, where we can edit our diagram in a graphical way. One can do all the editing functionalities like modeling new classifiers, zooming, spanning, modifying etc in this pane.

The Details Pane allows us to edit various details of our model. This includes the properties of each classifier, editing constraints, tagged values and checklist. This also simultaneously displays the source code generated for the respective elements modeled. The To-Do Pane displays the items on the models to-do list in a tree which sorts the list in a number of different ways. A drop down selection box at the top of the pane determines the layout of the tree.

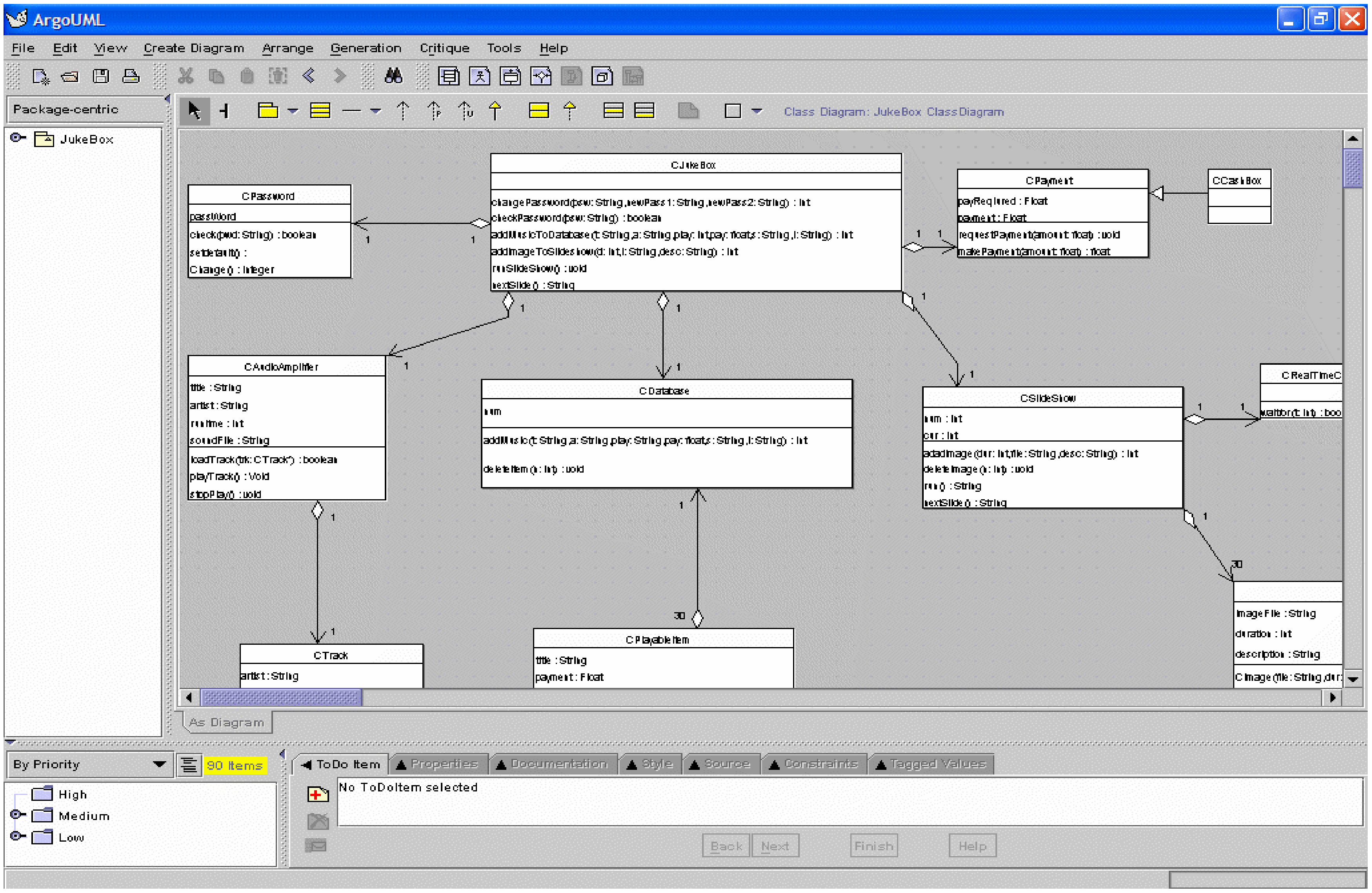


Figure 4.3 Workspace of ArgoUML

4.6 MasterCraft

4.6.1 Flexibility

MasterCraft supports Role Based Development. The various roles cover the entire Software Development Life Cycle (SDLC). Tasks that are logically related, tasks are grouped together and assigned to these roles. Roles supported in MasterCraft consist of two major roles: Managerial roles and User roles. These are further divided as shown below:

Managerial Roles

- Application Administrator
- Model Manager
- Construction Manager
- Version Manager

User Roles

- Analysis Modeler
- Design Modeler
- Construction Programmer [46]

Each of the roles is described below. A user can be assigned to more than one role which can be decided by the application administrator.

The Application Administrator can perform the following activities:

- Create users
- Create components
- Allocate users to components and roles

- Selecting the development environment installation type
- Define file server share
- Allocate file server-shares to application-specific standard servers
- Allocate server-shares to component-specific standard servers
- Generate report for server allocation and versioning activities
- Generate report for user allocation
- Perform server-side set up
- Extract model from jars
- Import models developed using Rational Rose
- Import user-models available in XMI format
- Export MasterCraft-developed user-models in XMI format
- Reset modeling status
- Purge journal data
- Reset the Control Table contents [47]

The role of the Analysis Modeler is to enter the UML model, to draw the different UML diagrams.

The Design Modeler has the following roles:

- To enter the Database and GUI model
- Perform impact analysis
- Define the component interface and inter - component dependencies.

The roles of the Construction Manager can be defined as follows:

- Export the model into the common pool
- Generate code from the model
- Build and release component jars
- Configure standards checker

- Perform check for conformance to standards
- Assign classes to each construction programmer

The Construction Programmer's tasks include the following:

- Implement the code for business logic
- Implement the code for business rules
- Unit test the modeled business services
- Test Application screens
- Perform file-level version control

The version manager is configured only if a version managing tool is installed. The tasks of Version Manager are as follows:

- Performs application level version control
- Checks in, checks out and merges application versions

4.6.2 Navigation : MasterCraft

The main window of the analysis modeler is shown in Figure 4.4 below. Each role has a separate window with a few differences. The main window includes a toolbar, selection window, diagram window. The selection window has a list from which one can select the components and decide a function upon it. The diagram window can be used to model the different elements.

There are other panes like Error Window and Output Window for the managerial roles. Output window displays the whether a task was successfully executed, displaying the results. The error window displays the errors thrown, displaying the type of error and a suggested in solution.

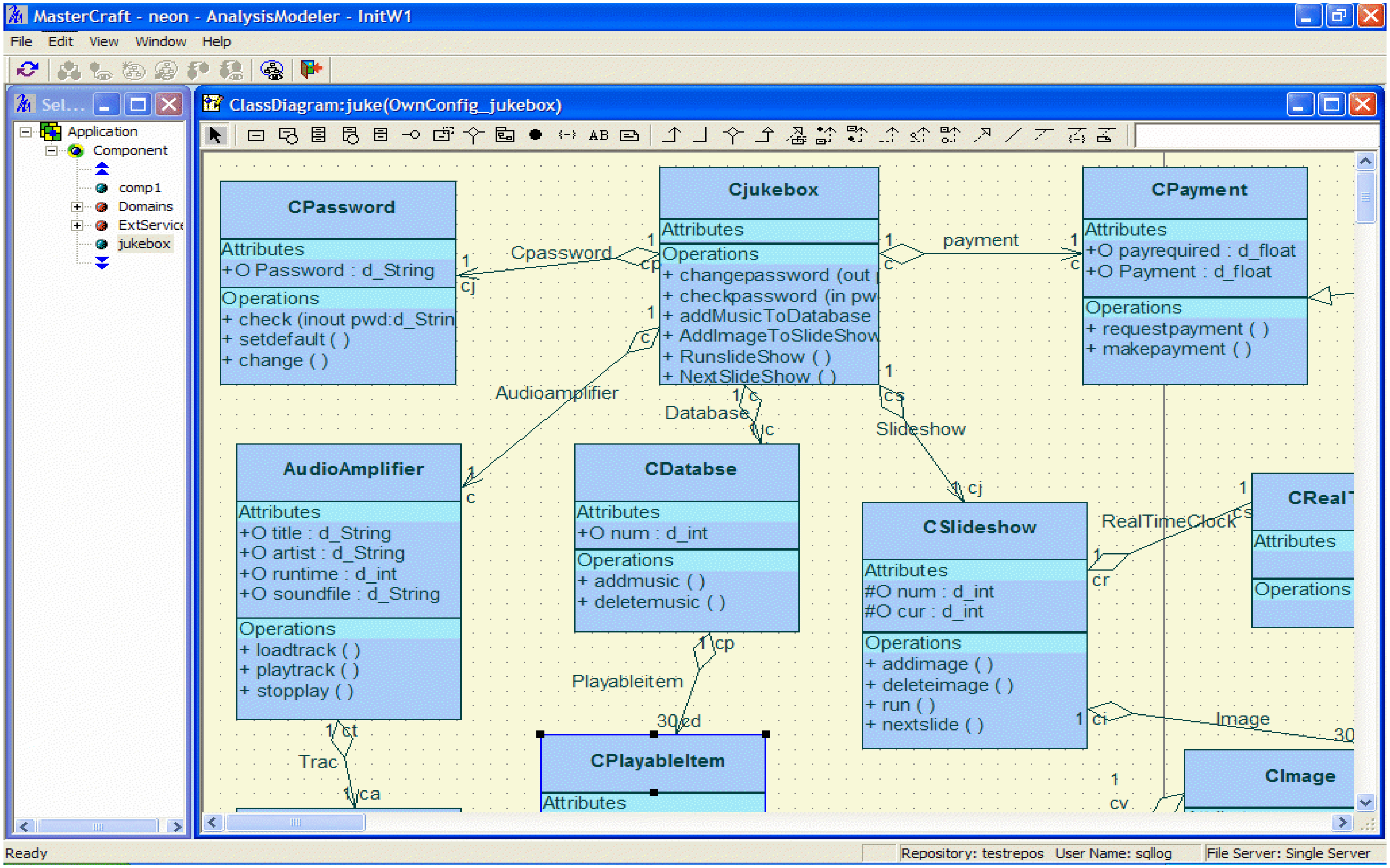


Figure 4.4 Workspace of MasterCraft

4.7 View of class diagram using each tool

The UML notation is rich and full bodied. It is comprised of two major subdivisions. There is a notation for modeling the static elements of a design such as classes, attributes, and relationships. There is also a notation for modeling the dynamic elements of a design such as objects, messages, and finite state machines. Static models are presented in diagrams called: Class Diagrams.

The purpose of a class diagram is to depict the classes within a model. In an object oriented application, classes have attributes (member variables), operations (member functions) and relationships with other classes. The UML class diagram can depict all these things quite easily. The fundamental element of the class diagram is an icon that represents a class.

This section presents the area of modeling class diagrams with the different tools along with other aspects like documentation and exporting diagrams.

Together Control Centre

Class diagrams can be exported as .gif, .wmf or .svg format. Figure 4.5 shows the class diagram for the jukebox application. Web documentation can be generated for the diagrams drawn.

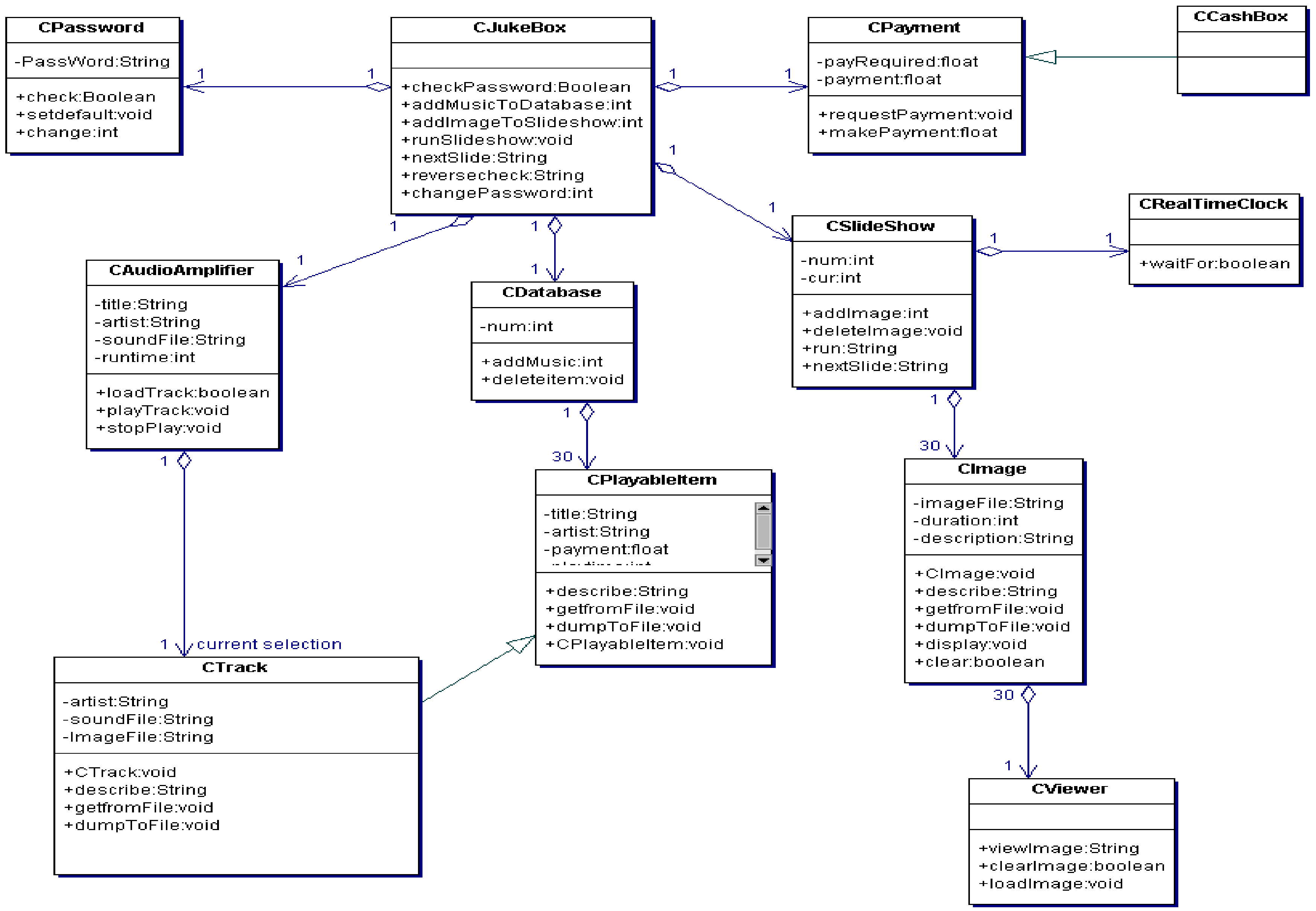


Figure 4.5 Class diagram in Together Control Center

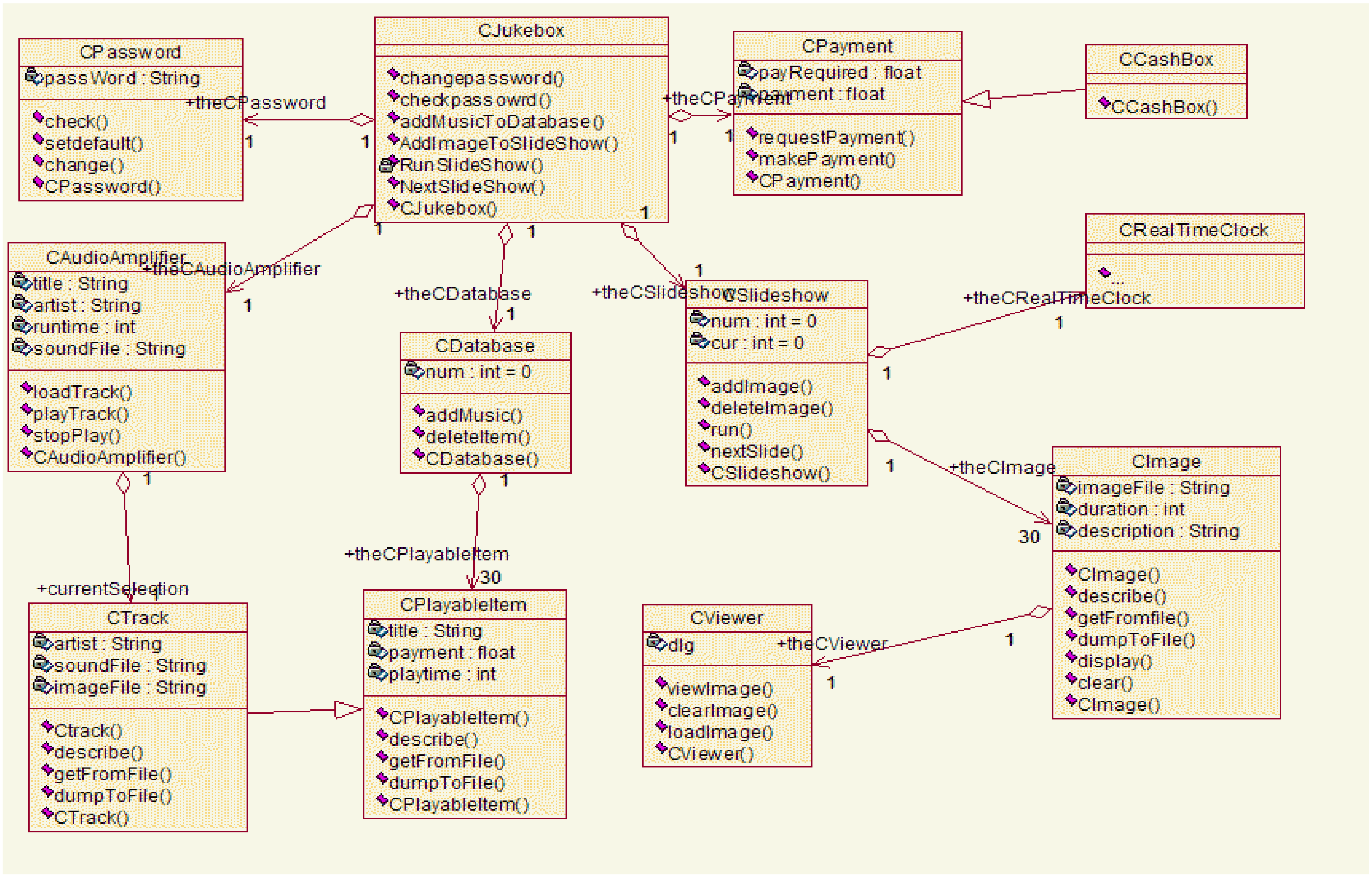


Figure 4.6 Class diagram in Rational Rose

Rational Rose

Figure 4.6 illustrates the class diagram in Rational Rose. Diagrams can be exported and saved as .jpeg, .bmp, .svg files. Web documentation in rational produces a clear separate documentation for each diagram and and the different classifiers in the diagram. Sequence diagrams for the use–case “AdminLogin” is shown in Figure 1.7

ArgoUML

ArgoUML helps to export a class diagram as .jpeg, .svg, post scripts, .eps. The class diagrams for the application Jukebox is illustrated in the Figure4.7.

MasterCraft

Figure 4.8 shows the class diagram for the application jukebox. MasterCraft export diagrams to rational rose and with a detailed diagram. The diagrams and individual elements in the diagram can be saved .bmp, .emf, and .gif files. An example of sequence diagram is shown in figure 4.9. The navigation through the diagram is not very flexible.

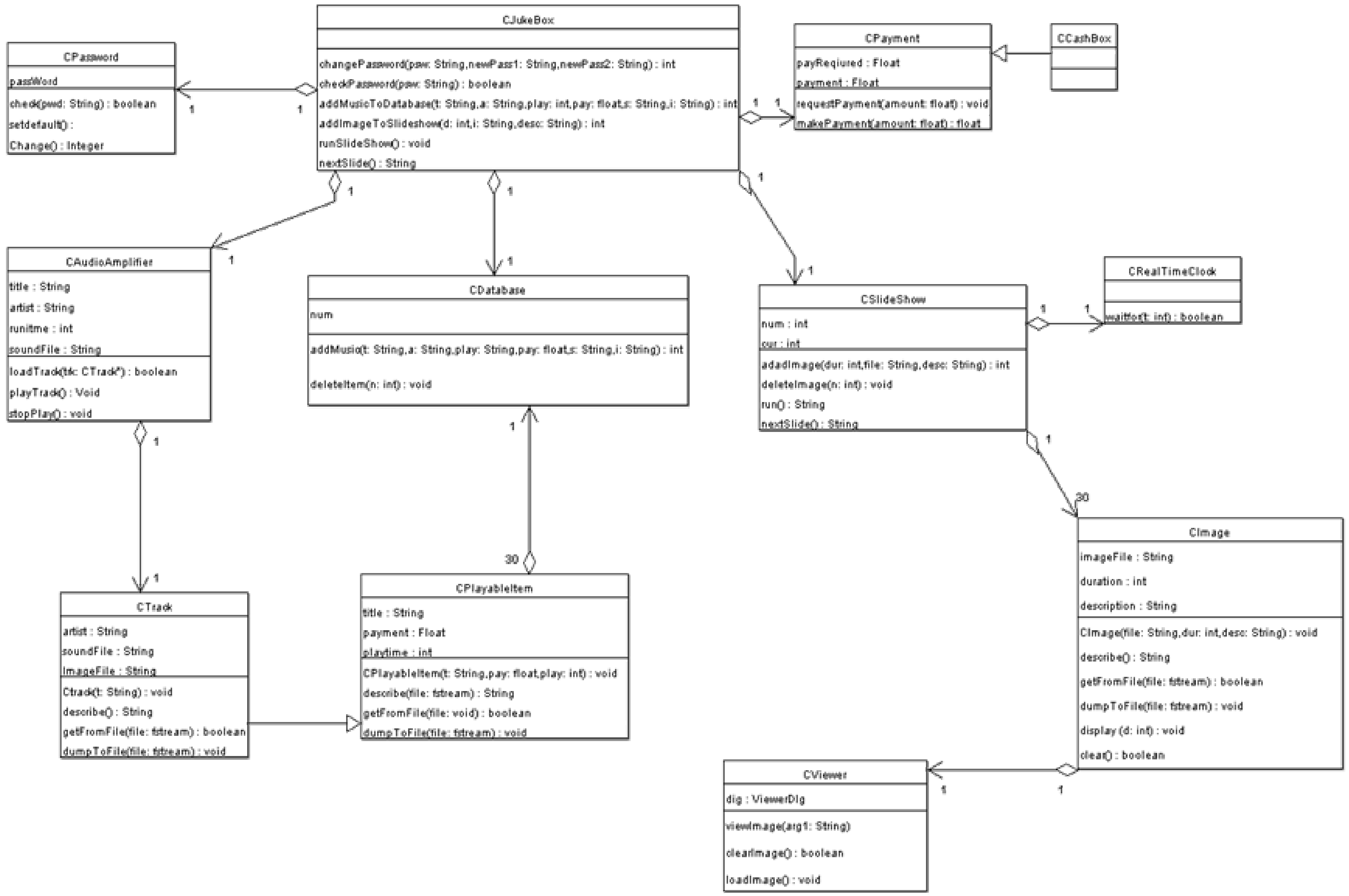

CPassword
passWord
check(pwd: String) : boolean
setdefault() :
Change() : Integer
CJukeBox
changePassword(psw: String,newPass1: String,newPass2: String) : int
checkPassword(psw: String) : boolean
addMusicToDatabase(t: String,a: String,play: int,pay: float,s: String,i: String) : int
addImageToSlideshow(d: int,i: String,desc: String) : int
runSlideShow() : void
nextSlide() : String
CPayment
payRequired : Float
payment : Float
requestPayment(amount: float) : void
makePayment(amount: float) : float
CCashBox
CAudioAmplifier
title : String
artist : String
runitme : int
soundFile : String
loadTrack(trk: CTrack*) : boolean
playTrack() : Void
stopPlay() : void
CDatabase
num
addMusic(t: String,a: String,play: String,pay: float,s: String,i: String) : int
deleteItem(n: int) : void
CSlideShow
num : int
cur : int
addImage(dur: int,file: String,desc: String) : int
deleteImage(n: int) : void
run() : String
nextSlide() : String
CRealTimeClock
waitfor(t: int) : boolean
CImage
imageFile : String
duration : int
description : String
CImage(file: String,dur: int,desc: String) : void
describe() : String
getFromFile(file: fstream) : boolean
dumpToFile(file: fstream) : void
display (d: int) : void
clear() : boolean
CTrack
artist : String
soundFile : String
ImageFile : String
Ctrack(t: String) : void
describe() : String
getFromFile(file: fstream) : boolean
dumpToFile(file: fstream) : void
CPlayableItem
title : String
payment : Float
playtime : int
CPlayableItem(t: String,pay: float,play: int) : void
describe(file: fstream) : String
getFromFile(file: void) : boolean
dumpToFile(file: fstream) : void
CViewer
dlg : ViewerDlg
viewImage(arg1: String)
clearImage() : boolean
loadImage() : void
1
30

Figure 4.7 Class diagram in ArgoUML

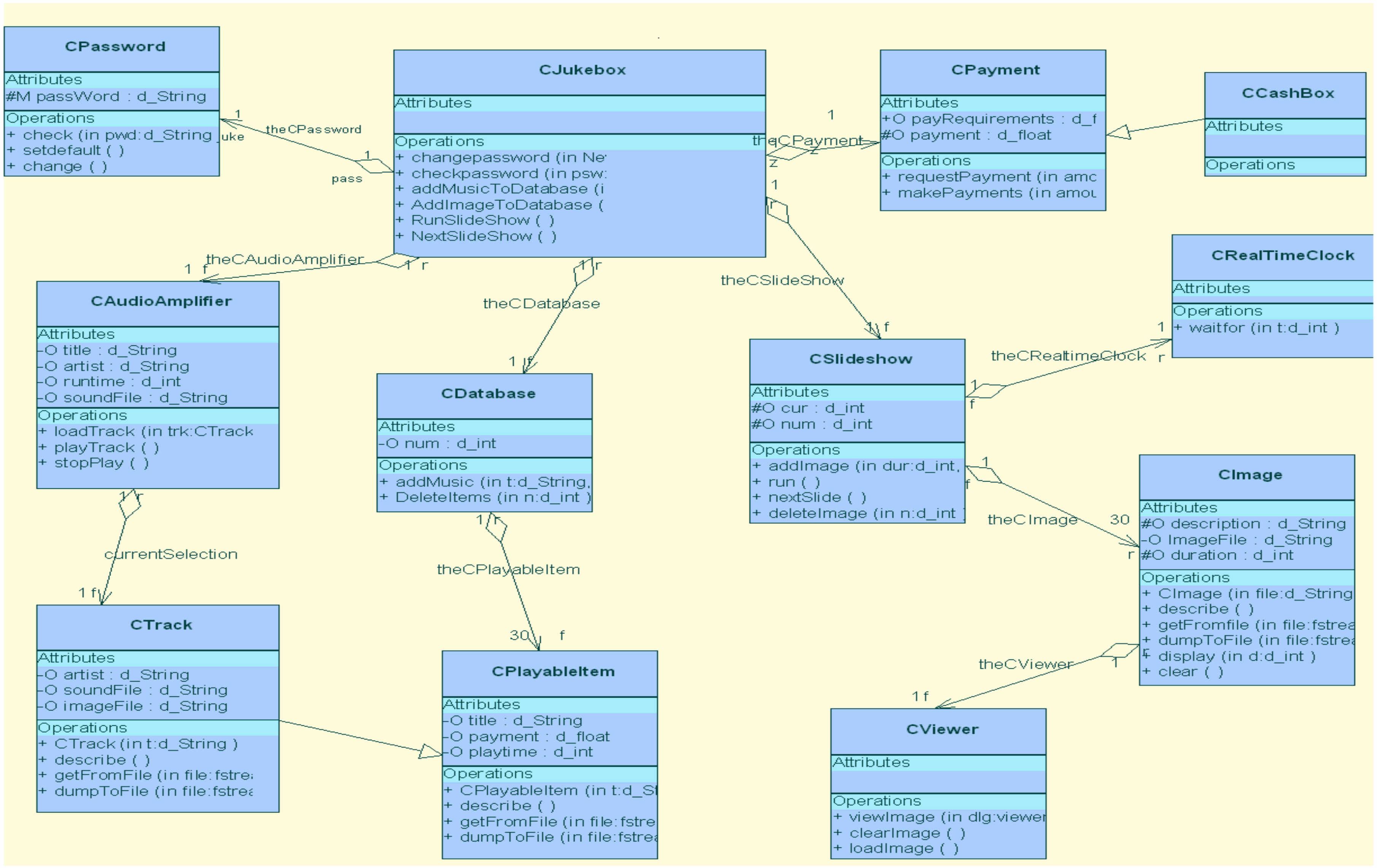


Figure 4.8 Class diagram in MasterCraft JAVA

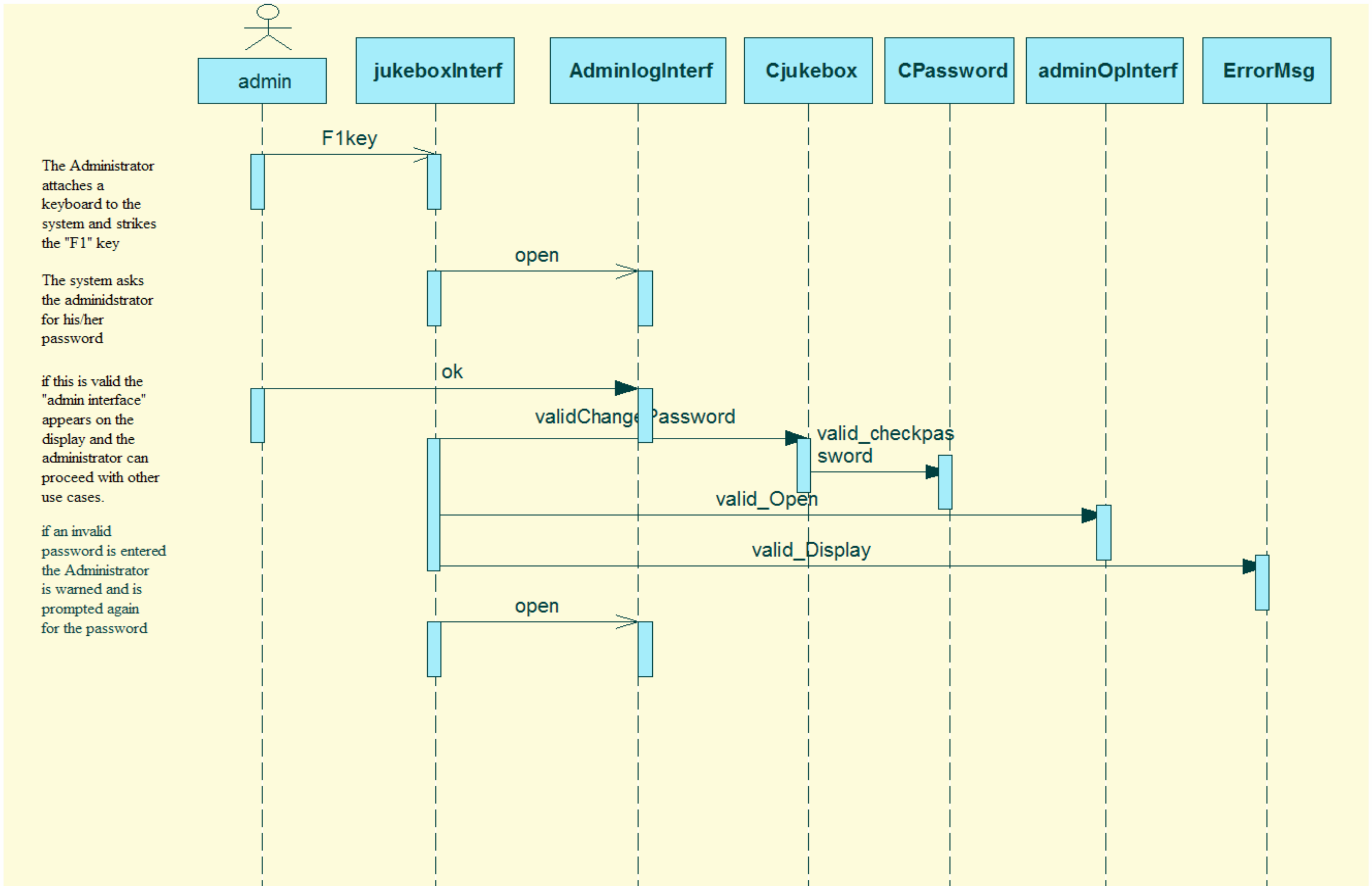

admin
jukeboxInterf
AdminlogInterf
Cjukebox
CPassword
adminOpInterf
ErrorMsg
F1key
open
ok
validChangePassword
valid_checkpassword
valid_Open
valid_Display
open
The Administrator attaches a keyboard to the system and strikes the "F1" key
The system asks the admindstrator for his/her password
if this is valid the "admin interface" appears on the display and the administrator can proceed with other use cases.
if an invalid password is entered the Administrator is warned and is prompted again for the password

Figure 4.9 Sequence diagram in MasterCraft for "AdminLogin"

4.8 Conclusion

The primary objective of a UML modelling tool is to model an Object Oriented application in using UML notation. Modelling can be done efficiently by drawing the design quickly and checking it against the standards of UML notation. This section looks at how ArgoUML, MasterCraft, Rational Rose and Together approach these issues. The interfaces for each tool are discussed. Sequence diagrams and Class diagrams are also highlighted in this section. ArgoUML and MasterCraft are not yet up to the mark when compared with the later two tools.

List of Figures

APPENDIX V

UML TOOLS – EVALUATION OF IMPLEMENTATION

Table of Contents

5.1 Introduction

Tools have always been an essential factor for humans to simplify his creative thoughts. Most of the time UML tools are used for designing the software. Since there are many tools now in the market each one claiming to provide certain degree of sophistication, it is up to us as users to analyze and evaluate these tools, comparing it to check if it meets up to the our business needs.

Round-trip engineering is any combination of multiple code generation and/or reverse engineering operations, though it most commonly refers to a series of operations alternating between the two. Code generation and reverse engineering are some of the key factors discussed in this section. Here we also discussion about how the code is generated by the different tools.

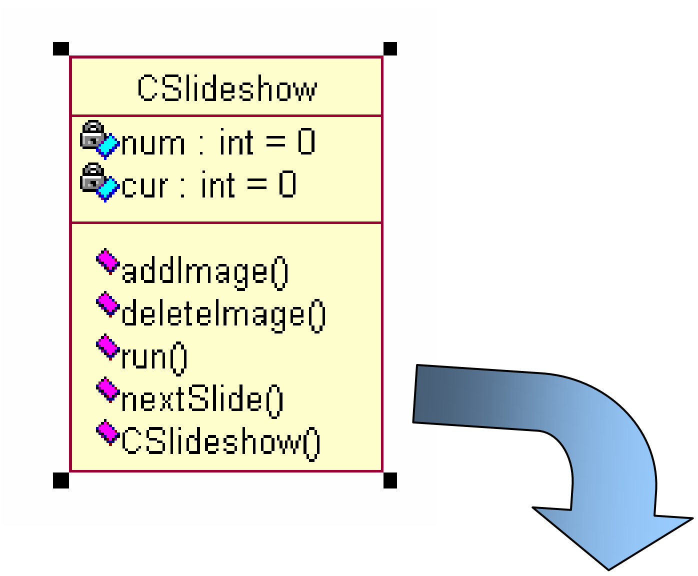
5.2 Code generation by each tool

Tools are used in order to simplify the work of a programmer. This is achieved through the functionality of round trip engineering in a tool. Round trip engineering includes two major features. One is code generation and the other reverse engineering. I believe that any tool which claims to be a UML tool should at least provide the functionality of Code Generation.

Reverse engineering is one of the factors which puts the tool in the forefront. Reverse engineering is to generate code from the code implemented or from the modified code. Having an in built editor helps the user from the trouble of finding an editor and integrating it with the tool. Though some tools provide separate editors with the facility to integrate with a UML tool.

Reverse engineering plays a major role in the maintenance of the software. The reverse engineering feature can be applied to existing code as well as code that are being developed. A tool should reverse engineer source code, build a model around existing code or restoring a model from archived files.

The code generation done by the tool generates a skeletal code for the classes, attributes, relationship between the classes and the parameters of the operations. This brings down the physical implementation time of the software by around 35 - 40 percentage.



```
//Source file: C:\\jukebox\\CSlideshow.java

public class CSlideshow
{
   private int num = 0;
   private int cur = 0;
   public CRealTimeClock theCRealTimeClock;
   public CImage theCImage[];

   /**
    * @roseuid 3F7C5A830358
    */
   public CSlideshow()
   {

   }

   /**
    * @param dur
    * @param file
    * @param desc
    * @return int
    * @roseuid 3F7B26ED02E9
    */
   public int addImage(int dur, String file, String desc)
   {
    return 0;
   }
```

Figure 5.1 Code generated for a class “Slideshow”

5.3 Implementation: Together Control Center

5.3.1 Features Of Editor

Together comes with a built-in full-featured text editor that allows you to work with any of the supported languages. The editor is flexibly configurable. You can configure the Editor using the Options dialog at any of the multiple configuration levels. The Editor can be tuned for working with different contents like: plain text, Java, C++, Visual Basic, IDL, HTML, XML, JSP, C#, Visual Basic .Net.

Together provides two types of bookmarks: global bookmarks that apply to the entire project, and local bookmarks that are used only within the currently opened file. [43] One can set the global bookmarks in the source code files and navigate to them from any open file that is part of your project. You can view, edit, classify, and navigate to bookmarks in the project using the Edit Bookmarks dialog, which is available on the right-click menu of the current line. Local bookmarks are fast and handy to operate. They are numeric instead of titled, meaning that they are numbered from 1 to 10. For this reason, there can be only ten local bookmarks per file.

Breakpoints specify where to stop code execution during debugging to permit inspection of variables, expressions, class members, and so on. This feature of the integrated debugger is accessible from the Editor.
The Editor Toolbar provides a number of buttons that significantly speed up the coding process. These buttons are: [43]

Code Sense, Advanced Code Sense: These features automatically complete your code.

Parameters Tool Tip: This feature is also a part of code completion, and displays the list of possible parameters for the method under the cursor.

Surround With:

Surrounds the selected lines of code with one of the specified constructions (for, if-else, try-catch).

Toggle Comments:

This feature enables you toggle between commenting and uncommenting the selected section of code.

Override/Implement Methods:

Enables you to choose methods of the parent class to be implemented or overridden.

Expand Snippet:

Expand Snippet expands one of the pre-defined snippets into the code block that it represents.

Browse Symbol:

Browse symbol enables you to view the source code of the library classes.

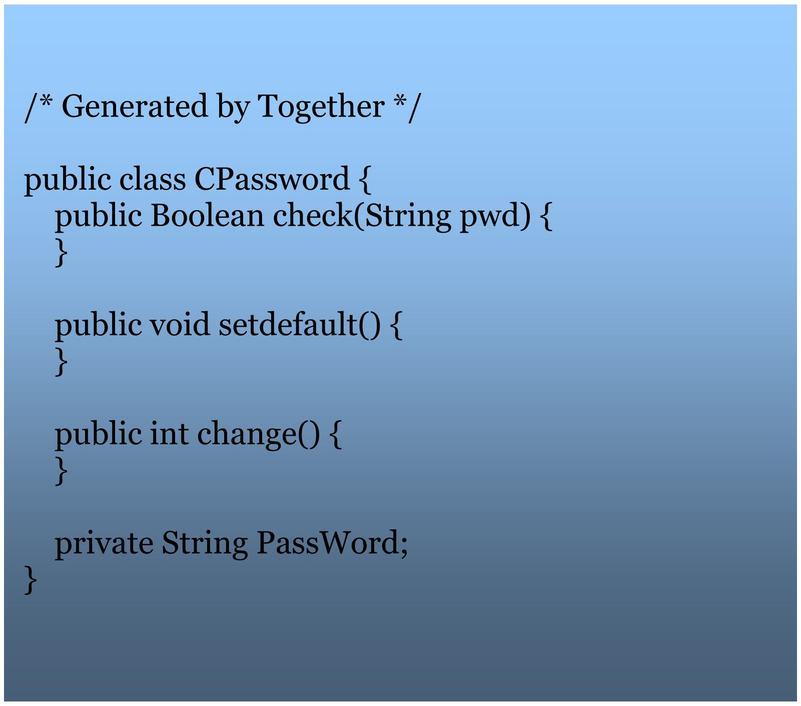
Previous / Next Declaration:

This feature Navigates through the list of declarations within the current class.

5.3.2 Forward Engineering

The figure below [Figure 5.2] gives a snapshot of the code generated by together control center. It is very plain and simple without any comments. There are no comments directing the programmer where to insert his code.

The code generated does not include default constructors or destructors. The operations are generated on top and the attributes and the relations are generated at the end.



```
/* Generated by Together */

public class CPassword {
   public Boolean check(String pwd) {
   }

   public void setdefault() {
   }

   public int change() {
   }

   private String PassWord;
}
```

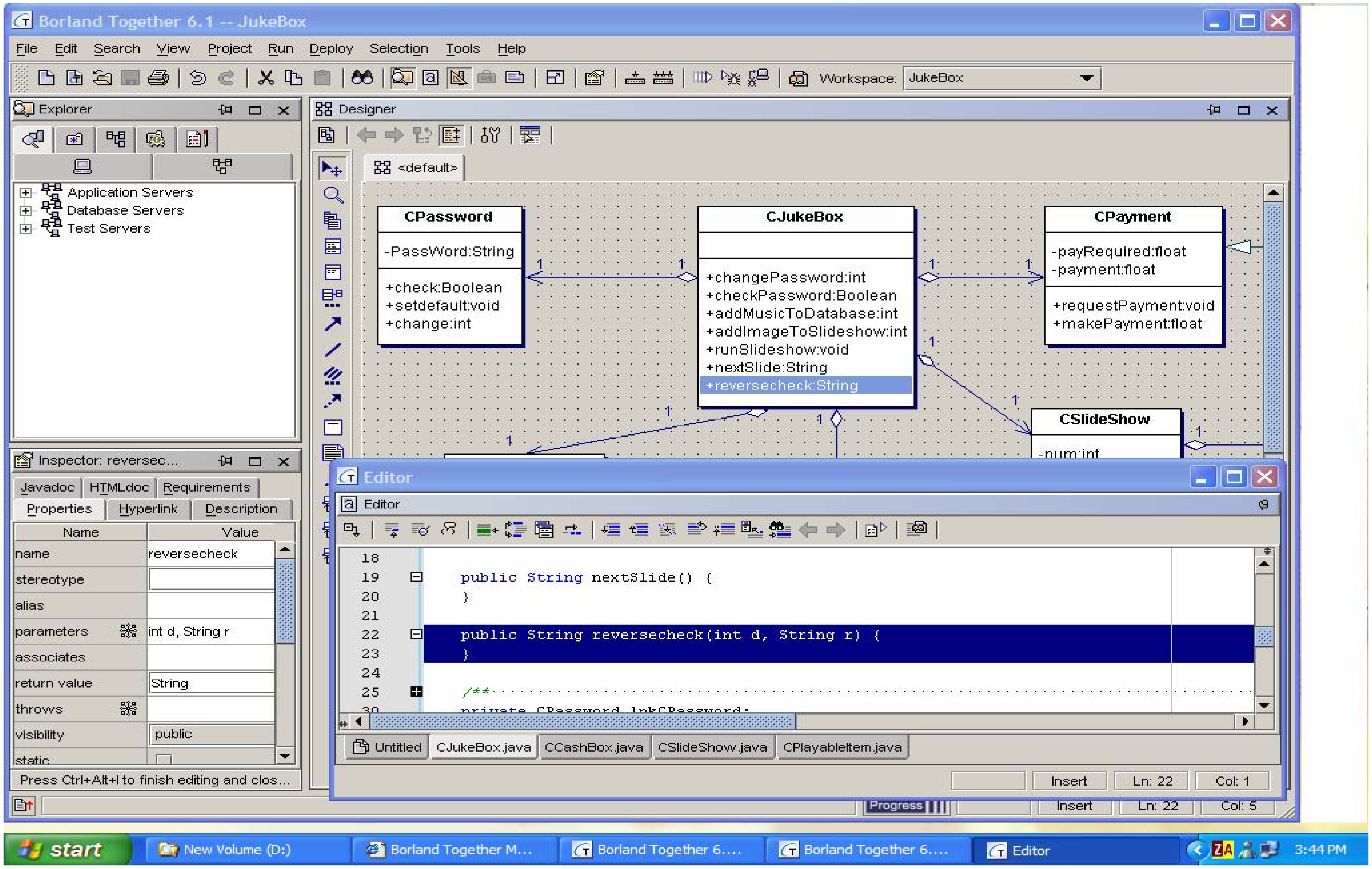
Figure 5.2 Code generated by Together for class Password

5.3.3 Reverse Engineering

A central feature of Together is Live Source. This is the ability of the tool to immediately synchronize class diagrams with the implementation code. Live Source means that your UML class diagrams are always synchronized to the source code that implements them. When you change a class diagram, Together immediately updates the corresponding source code. When you change the code, Together updates the visual model. There is no intermediary repository, no batch code generation. The Live Source feature applies to existing code as well as the code that are being developed.

Together can do reverse engineering to the source code. It builds a model around existing code and can restore a model from archived files.

The figure below [Figure 5.3] gives a view of how Together achieves synchronization with an integrated editor. In the example below the new operation “reverse check” was added to the class *CJubebox,* in the editor. This is immediately shown in the model as a new operation Reverse Check. This new operation added is highlighted in figure both in the editor pane and in the designer pane.



Name	Value
name	reversecheck
stereotype	
alias	
parameters	int d, String r
associates	
return value	String
throws	
visibility	public
static	

```
    public String nextSlide() {
    }

    public String reversecheck(int d, String r) {
    }

    /**...
    private CPassword lnkCPassword;
```

Figure 5.3 Reverse Engineering with Synchronised Editor - Together

5.4 Implementation: Rational Rose

5.4.1 Forward Engineering

Code generation (also called forward engineering) is the process of generating Java source from one or more classes in a Rose model. Round-trip engineering is any combination of multiple code generation and/or reverse engineering operations, though it most commonly refers to a series of operations alternating between the two. Model and source code are kept synchronized over an extended period of time and through multiple changes. If you change the model, use a code generation operation to make the corresponding changes to the code. If you change the code, use reverse engineering to make the corresponding changes to the model.

Forward engineering in Rose is component-centered. This means that the Java source generation is based on the component specification rather than on the class specification. To do this, you create a class and then assign it to a valid Java component. Or, Rose creates the component for you when your model's default notation is Java.

When you forward engineer a Java model element, its characteristics are mapped to a corresponding Java-language construct. For example, a Rose class forward-engineers, through its component, to a .java file; a Rose package forward-engineers to a Java package, and so on. In addition, when you forward engineer a package, a .java file is generated for each component belonging to the package. Each .java file contains the definitions for any classes assigned to that component.

Rose Java offers an auto-synchronization mode that automatically initiates code generation any time you create or modify any Java element in your model. By default this is off but one can enable this feature through the Java Project Specification.

Because Auto Synchronization is normally off, Rose generates RoseIDs for Java methods. This feature allows Rose to track method name changes in the code. When Auto Synchronization is turned on, Generate Rose ID should be turned off (on the Code Generation tab of the Project Specification). The RoseID is a Java comment that takes the form:

```
@roseuid <string>
```

Model ID comments of the form:

```
//##ModelId=392B160E0157
```

may be inserted into the code during code generation or reverse engineering. These comments help Rose match declarations in the code with the corresponding elements in the model when doing both code generation and reverse engineering. The figure 5.4 shows the code generated by Rose in Java for the class CPassword.

```
//Source file: C:\\jukebox\\CPassword.java


public class CPassword
{
  private String passWord;

  /**
   * @roseuid 3F7945060322
   */
  public CPassword()
  {

  }

  /**
   * @param pwd
   * @return Boolean
   * @roseuid 3F793A880208
   */
  public Boolean check(string pwd)
  {
   return null;
  }

  /**
   * @return Void
   * @roseuid 3F793A9701D7
   */
  public Void setdefault()
  {
   return null;
  }

  /**
   * @return Integer
   * @roseuid 3F793AE2027F
   */
  public Integer change()
  {
   return null;
  }
}
```

Figure 5.4 Code generated by Rational Rose for class Password

Rose specifications enable you to document your model elements by adding text to various Documentation fields. Rose Java uses the text you supply to create Javadoc tagged comments in the code it generates. There are two comment types that Rose generates:

- **Asterisk Style**. The Asterisk style inserts an asterisk at the start of the comment. This is the standard Java style.

- **Javadoc style**. This style uses Javadoc tags that the Javadoc compiler uses to create HTML pages that describe various Java constructs such as classes, interfaces, constructors, methods, etc

5.4.2 Reverse Engineering

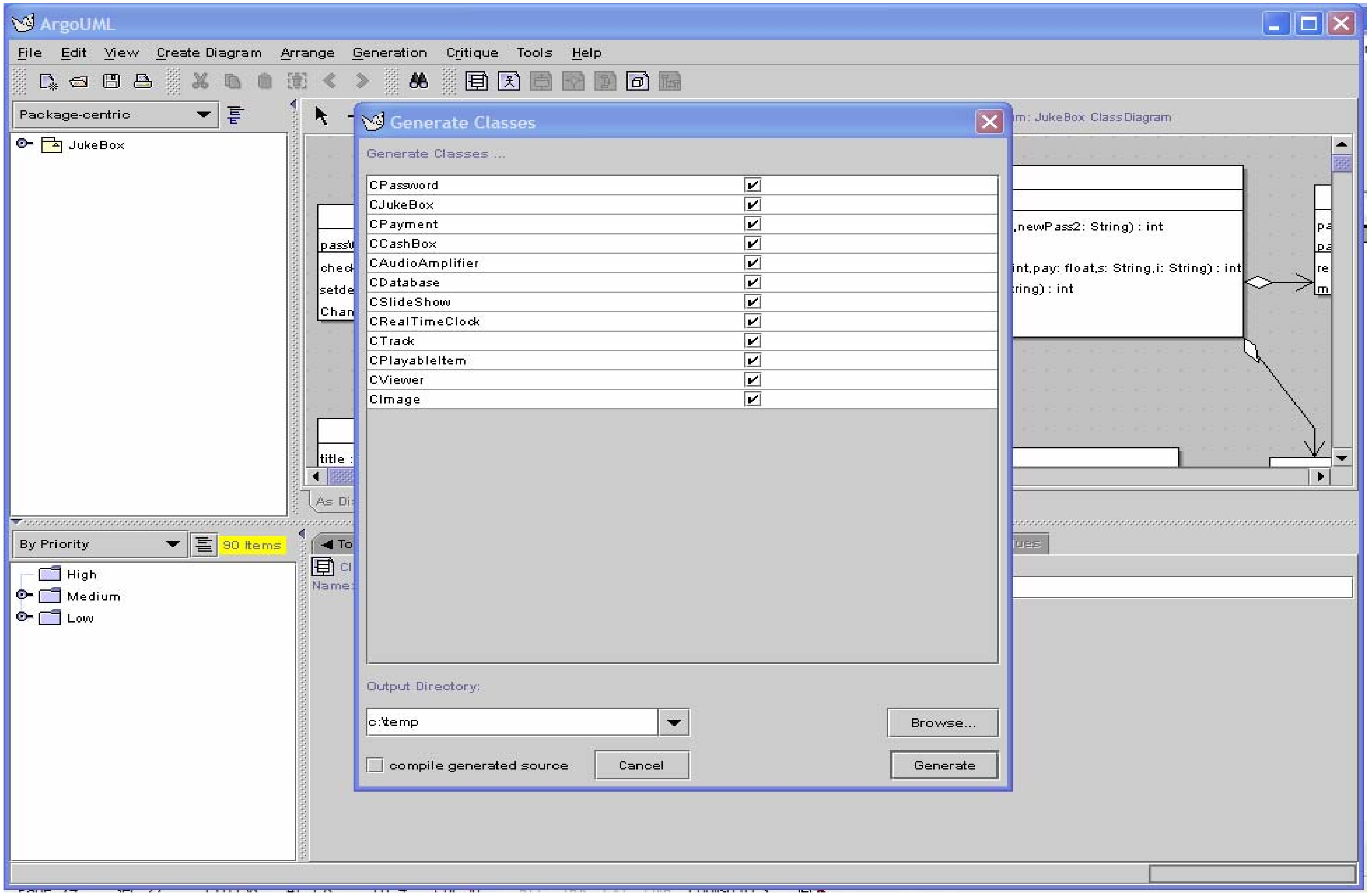
Rose Java reverse-engineers a .java file as a Java-language component in your model. This component includes the package information that locates the file in you directory structure. At the same time, it creates the class(es) contained in the .java file. You can view reverse-engineered classes and components in the Rose Browser. Rose does not automatically create class or component diagrams based on newly reverse-engineered classes. To add classes or components to diagrams you can drag and drop them from the browser into new or existing diagrams

Do not change both the model and the code at the same time since it is difficult to get them synchronized again. If the changes in the model and the code are in different classes, then it may be possible to selectively do code generation and reverse engineering on a class-by-class basis. If the same class is changed in both the model and the code, one set of changes will have to be overwritten in order to synchronize that class again.

5.5 Code generation: ArgoUML

Code generation in ArgoUML is very simple to do. In the workspace of ArgoUML there is "Generate" on the toolbar menu. This is to generate code. You have the option to generate code for Selected Classes or All the Classes. If All the Classes is selected the "Generate Classes" window appears [Figure 5.5]. From here one can select the classes for which code generation is done.

The source code can be viewed along with the model since it is synchronized with the model. So any change to the model is immediately reflected. The code generated by ArgoUML is shown in Figure 5.6. It is simple without any comments. The attributes and relations are placed first then the operations are placed. ArgoUML does not cater for reverse engineering.



Generate Classes ...

CPassword	✓
CJukeBox	✓
CPayment	✓
CCashBox	✓
CAudioAmplifier	✓
CDatabase	✓
CSlideShow	✓
CRealTimeClock	✓
CTrack	✓
CPlayableItem	✓
CViewer	✓
CImage	✓

Output Directory: c:\temp

compile generated source [ ]

Figure 5.5 Process of Code Generation in ArgoUML

```
import java.lang.String;
import java.lang.Integer;

public class CPassword {

public String passWord;
  /* {transient=false, volatile=false}*/


public boolean check(String pwd) {
  return false;
  }
public void setdefault() {
  }
public Integer Change() {
  return null;
  }
}
```

Figure 5.6 Code generated by ArgoUML for class Password

5.6 Code generated: MasterCraft

5.6.1 Role of Construction Manager in Implementation

The construction manager's role is responsible for initiating the construction activity for the application and for performing building jars of the components and deploying the individual group of components – a component and its supplier-components. This role is also responsible for managing and monitoring the construction activity being carried out for the entire application. Some of the functions done by the Construction Manager are:

- Generate code for the component
- Manage user-workspaces for the ConstructionProgrammers
- Check conformance of the code to the Java standards
- Create application tables
- Prepare application for deployment

Code for a component is generated from the object model of the component that is available as a CDIF file. User-model of the selected component needs to be translated into a standard format that is easily understood by the other tools in MasterCraft. The model in the relational database gets translated to the one in a flat database. The code generators in MasterCraft need a Case Data Interchange Format (CDIF) file of the component as the input.
The following files are generated:

- Object model files
- Domain description files
- Java files for the classes
- Java files for interface of the classes
- Java files for the WinClasses
- SQL files for queries
- Java files for Batch Programs
- Java files for Batch Functions
- Unit-testing driver Java files
- Input files

- Makefiles
- DDLs
- Rulebase
- Test data specification templates for Test Data Generator [66]

All the Java files for classes, WinClasses, and interfaces are compiled during the 'Generated Code' action to create a stub jar. A stub jar is the jar with dummy implementations. After generating code, the Construction Programmer can pick up the class templates and code them. These classes can then be compiled, built and unit tested.

5.6.2 Role of construction Programmer in Implementation

The Construction Programmer's role is used for performing all the coding and unit-testing actions after the Construction Manager has completed the initial set-up and code generation. The Construction Programmer can edit the generated code template to enter the business logic. Typically no changes should be made to the MasterCraft generated code. However, sometimes you may find that there are some model elements that may not have been modeled and need to be created. These can be created and used while coding, for example, you may add an attribute to a class or a parameter to an operation or service. These elements can thus be introduced in the construction phase. However, it is then required that the change be validated and incorporated in the model from which the code was generated. After understanding the need of the code changes, the Construction Manager ascertains the correctness of changes in the code, and approves the updating of the model accordingly during 'Synchronize User-workspace'.

```
// ## MasterCraft generated code starts here
// ## This code should not be changed
package comp1;

import Domains.*;
import ErrorMessages.*;
import com.tcs.mastercraft.mctype.*;
import com.tcs.mastercraft.mctype.errlib.*;
import com.tcs.mastercraft.mcutil.* ;
import java.util.*;
import java.io.*;
import java.text.*;
import java.sql.*;
import com.tcs.mastercraft.mcdiagnosis.*;
// ## User-required import statements could start here onwards


// ## End of User-required import statements

/**
*/
class CPassword extends MasterCraftObject implements java.io.Serializable
{
        // Following are the Class Attributes
        /**
         */
        protected StringBuffer passWord; // Domain d_String maps to StringBuffer
here

        private MasterCraftBitSet specFlag;


                // ## User-required attribute declaration could start here onwards


                // ## End of User-required attribute declarations
// This is a Default Constructor
        CPassword ( )
        {
                passWord = new StringBuffer(256);
                specFlag = new MasterCraftBitSet(1);
        }

        // DeepCopy Constructor
        public void deepcopy(Object inst_CPassword) throws Exception
        {
                if ( inst_CPassword instanceof  CPassword )
                {
                        CPassword __CPassword_obj = ( CPassword ) inst_CPassword;

                        passWord.replace( 0, __CPassword_obj.passWord.capacity(),
__CPassword_obj.passWord.toString() );
                        specFlag.deepcopy( __CPassword_obj.specFlag );
                }
        }

        public int getId()
        {
                return 28;
        }

        public void SetspecFlag( MasterCraftBitSet spec ) throws Exception
        {
                specFlag.deepcopy(spec);
        }

        public MasterCraftBitSet GetspecFlag( )
        {
                return specFlag ;
        }
```

```
public int IsSpecifiedpassWord()
        {
                if( specFlag.isBitOn( 0 ) == true )
                {
                        return 1;
                }
                else
                {
                        return 0;
                }
        }

        public void SetpassWord (final StringBuffer iStringBuffer ) throws Exception
        {
                passWord.replace( 0 , passWord.length() , iStringBuffer.toString() ) ;
                specFlag.setBit( 0 );
        }

        public void UnmarkpassWord ()
        {
                specFlag.resetBit( 0 );
        }

        public StringBuffer GetpassWord ()
        {
                return ( passWord );
        }

        public void UnmarkAll ()
        {
                specFlag.resetAll();
        }
        public String toString ( )
        {
                String s = "";
                s = s + "\nClass CPassword : " ;
                if ( IsSpecifiedpassWord() != 0)
                {
                        s = s + "\npassWord = " + passWord + " (StringBuffer) ";
                }
                else
                {
                        s = s + "\npassWord = _NS_ " + " (StringBuffer) ";
                }
                s = s + "\nspecFlag = " + specFlag.toString() + " (MasterCraftBitSet) "  ;
                s = s + "\nEND OF CPassword \n\n" ;
                return s;
        }
// ## End of MasterCraft generated code



        // ## Signature of check_30 method should not be changed
        /**
          * @param pwd
         */
        public  void check ( StringBuffer pwd  ) throws Exception
        {
                String signature = new String("CPassword::check(StringBuffer):void");
                ServerContext.SetretError( new ErrorType() );
                // ## Add your code for check_30 method here




        }
        // ## End of check_30 method


        // ## Signature of setdefault_36 method should not be changed
        /**
         */
```

```
public  void setdefault ( ) throws Exception
        {
                String signature = new String("CPassword::setdefault():void");
                ServerContext.SetretError( new ErrorType() );
                // ## Add your code for setdefault_36 method here





        }
        // ## End of setdefault_36 method


        // ## Signature of change_37 method should not be changed
        /**
         */
        public  void change ( ) throws Exception
        {
                String signature = new String("CPassword::change():void");
                ServerContext.SetretError( new ErrorType() );
                // ## Add your code for change_37 method here





        }
        // ## End of change_37 method
        // ## User-required operation declaration could start here onwards

        // Sample format to add an operation
        // public void ExampleOperation( int Param1, StringBuffer Param2 )
        // {
                //              CODE_START


                //              CODE_END
        // }

        // ## End of User-required operation declarations

}
```

Figure 5.7Code generated by MasterCraft Together for class Password

Though the generated code is long and appears to be complicated, it is well commented. Figure 5.7 shows the code generated for the class Password. The business logic for the operation Setdefault is implemented between the two lines shown below:

```
// ## Add your code for setdefault_36 method here

}
// ## End of setdefault_36 method
```

The tags between which you can add the required import statements are:

```
// ## User-required import statements could start here onwards
import xyz.*;
// ## End of User-required import statements
```

5.6.3 Q++ A Technology Independent Language

The business functionality of an application is independent of the implementation technology being used to realize it as a software system. Appreciation of this separation is critical for the application to be configurable with respect to the implementation technology. MasterCraft provides a high-level specification language, Q++, which is designed to achieve transparency in specifying the program logic, with respect to the technology being used in the application in a declarative manner. [67]

Q++ is a high-level specification language that is used to write the application logic on the server-side. It has the following features:

- Object-oriented and model-aware

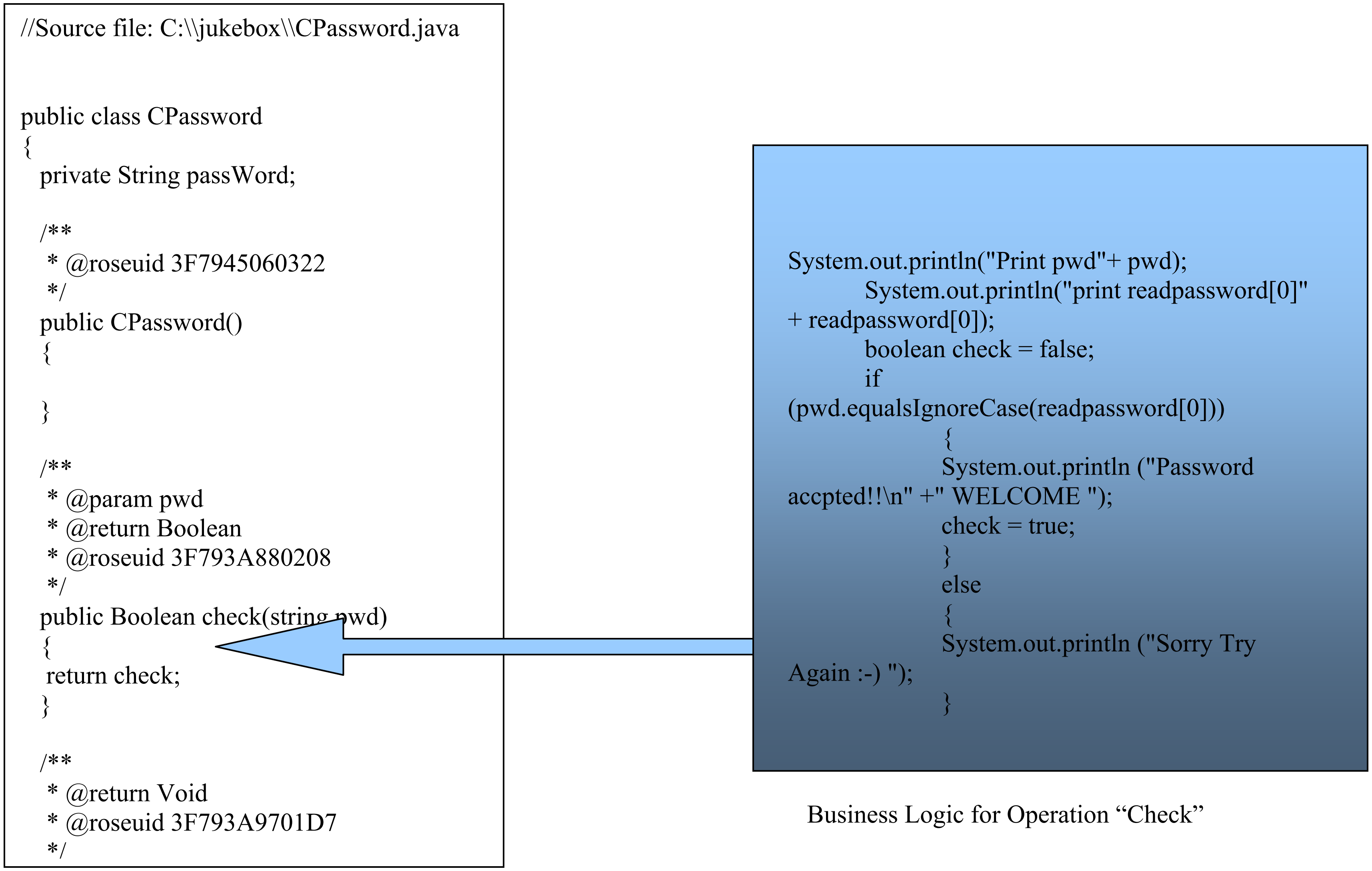
- Transparent memory management
- Comprehensive error handling
- Type checking across the network
- Simplified query-processing with multiple records
- A special data type called MULTIROW and its associated methods to handle multiple rows of data [64, 67]

5.6.4 Reverse Engineering

Extracting models from external jar is the term used for reverse engineering in MasterCraft. This includes extracting models from jars and model updating with the required code changes. This functionality allows reverse engineering up to a certain level but does not allow you to update the model. The later version of MasterCraft has incorporated this property.

5.7 Business Logic Implementation

In the above sections we discuss the code generation done by each tool. This code generated is merely a skeletal code for the classes, parameters and relationships. To have a workable code, a programmer has to plug-in the business logic for the different operations. He might also have to add in new classifiers during the implementation. The figure5.8 shows how the code is implemented between the specified lines of the generated code.



```
//Source file: C:\\jukebox\\CPassword.java


public class CPassword
{
  private String passWord;

  /**
   * @roseuid 3F7945060322
   */
  public CPassword()
  {

  }

  /**
   * @param pwd
   * @return Boolean
   * @roseuid 3F793A880208
   */
  public Boolean check(string pwd)
  {
   return check;
  }

  /**
   * @return Void
   * @roseuid 3F793A9701D7
   */
```

```
System.out.println("Print pwd"+ pwd);
            System.out.println("print readpassword[0]"
+ readpassword[0]);
            boolean check = false;
            if
(pwd.equalsIgnoreCase(readpassword[0]))
                        {
                        System.out.println ("Password
accpted!!\n" +" WELCOME ");
                        check = true;
                        }
                        else
                        {
                        System.out.println ("Sorry Try
Again :-) ");
                        }
```

Business Logic for Operation “Check”

Figure 5.8 Implementation: Plug –in the business logic

5.8 Conclusion

Tools are used to curtail time and facilitate the development of a product. In software the final product is a workable automated code. Tools help to develop these software products with quality and better return on investment. This section wraps up the functionalities like forward engineering, reverse engineering and integrated editor in each of the tool used for the evaluation. Code generation, reverse engineering and updating the model are some of the vital functionalities when maintaining a system. Keeping track of changes in the code and /or in the model and versioning it is also an important aspect. Above all a tool which is easy to use is always alluring to its user.

List of Figures

APPENDIX IV

EVALUATION RESULT

Evaluation Table

Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
Repository:					
database	**2**	1	2	1	3
Customization:					
Components and tools	**2**	3	1	2	1
HTML documentation:					
Generate Web Reports:	**3**	2	1	3	2
Save diagrams to include in reports	**3**	3	1	3	3
Usability:					
First Contact:	**2**	3	3	2	1
Ease of use:	**3**	2	3	3	2
Pick List:					
Pick list of the classes in the model while drawing.	**2**	3	1	1	1
Tree structure in a pane where all the classes can be viewed.	**3**	3	3	3	3
Select methods to draw sequence or	**3**	2	1	3	3

Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
collaboration diagram.					
Drag and drop	**3**	3	3	3	3
Interface Presentation					
Allows changing the font and size of labels on diagrams.	**3**	3	1	3	3
States are visibly distinct	**2**	3	2	2	2
Documentation and help files					
step by step documentation	**3**	3	2	3	2
Online help	**2**	3	1	3	1
Keyword search facilities	**3**	1	1	1	1
Short – cut keys	**1**	1	1	1	1
Printing Support:					
Fit diagram to a single page	**3**	3	3	3	2
Print preview	**3**	3	1	3	3
Scaling functionality	**3**	2	1	3	1
Exporting Diagrams:					
Save diagrams any picture editor format:	**3**	3	1	3	3
XMI format:	**2**	3	3	3	3
Robustness:					

Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
Does tool crash or corrupt diagrams:	**3**	2	3	3	3
Automatic saving at periodic intervals:	**1**	1	1	3	1
New Releases:					
Any New version releases announced:	**3**	3	1	2	3
Round Trip Engineering:					
Code generation:	**3**	3	3	3	3
Reverse engineering:	**3**	3	2	3	
Synchronization with editor:	**2**	1	3	3	2
Model navigation:					
Zooming and panning:	**3**	3	1	3	3
Navigating between source code and diagram:	**2**	1	2	3	
Diagram views:					
Customizing details of classes:	**3**	3	3	2	2
Reveal and hide methods:	**3**	2	1	2	2
Visibility of methods and attributes:	**3**	3	3	3	3
UML support:					
Key Notation parts					

Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
Use-case diagrams	**3**	2	3	3	3
Class diagrams	**3**	3	3	3	3
Sequence diagrams	**3**	3	1	3	3
Collaboration diagrams	**3**	3	3	3	3
State diagrams	**3**	3	3	3	3
Activity diagrams	**3**	3	3	3	3
Component diagrams	**3**	1	1	3	1
Deployment diagrams	**3**	1	3	3	3
Package diagrams	**3**	3	1	1	3
Class box size flexibility	**3**	3	3	2	2
Line flexibility	**2**	3	3	2	2
Independent placement of association end names	**3**	3	3	3	3
Independent placement of multiplicity labels	**3**	3	3	3	3
Preservation of position of end names and multiplicity labels	**3**	3	1	3	3
Distinguish between remove from diagram and delete from model	**3**	3	3	1	3

Features	***Weightings***	***Rational Rose***	***Argo UML***	***Together***	***Master Craft***
Restoration of relationships in new diagrams	**3**	3	1	3	2
N-ary associations	**3**	3	3	3	3
undo functionality for diagrams	**2**	1	1	3	3
UML profiles (stereotypes and tagged values)	**2**	3	1	2	2
Total Score		***352***	***279***	***359***	***328***