A COMPARATIVE STUDY OF THE UNIFIED MODELING LANGUAGE AND THE BUSINESS OBJECT NOTATION

A Thesis

Presented to

the Faculty of the Graduate School

Ateneo de Manila University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Paulina Heruningsih Prima Rosa

1999

The thesis entitled:

A COMPARATIVE STUDY OF THE UNIFIED MODELING LANGUAGE AND THE BUSINESS OBJECT NOTATION

submitted by Paulina Heruningsih Prima Rosa has been examined and is recommended for Oral Defense.

JOHN PAUL C. VERGARA, Ph.D.
Chairman

JOHN PAUL C. VERGARA, Ph.D.
Adviser

JAMES A. O'DONNELL, S.J., Ph.D.
Assistant Dean
Graduate School of Arts and Sciences

Comprehensive Examination Passed: 16 May 1998

The Faculty of the Graduate School of the Ateneo de Manila University accepts the thesis entitled:

A COMPARATIVE STUDY OF THE UNIFIED MODELING LANGUAGE AND THE BUSINESS OBJECT NOTATION

Submitted by Paulina Heruningsih Prima Rosa in partial fulfillment of the requirements for the degree of Master of Science in Computer Sciences, major in Software Engineering.

EMILIO C. VERGARA, III, MBM
Member

JESUS VICTOR A. FERIA, M.S.
Member

MA. REGINA E. ESTUAR, M.S.
Member

JOHN PAUL C. VERGARA, Ph.D.
Adviser

MARI-JO P. RUIZ, Ph.D.
Dean
Graduate School of Arts and Sciences

Grade : Very Good

Date : 20 April 1999

ACKNOWLEDGEMENT

This thesis would not have been accomplished without the support and help from so many people. To the following I am greatly indebted:

- Sanata Dharma Foundation and Sanata Dharma University for providing me the opportunity to study in ADMU.
- The Rector of Sanata Dharma University, Dr. M. Sastrapratedja, S.J., and the treasurer of Sanata Dharma Foundation, Fr. Opzeeland, S.J., for their support and understanding.
- The Dean of the Faculty of Mathematics and Natural Sciences of Sanata Dharma University, Dr. Frans Susilo, S.J., for constantly being so trusting and supportive throughout my study.
- My adviser, Dr. John Paul Vergara, for his constant guidance and direction throughout the writing of this thesis. Thank you for your assurance that helped me accomplish all the phases.
- My panelists, Mr. Jesus V. Feria, Mr. Emilio C. Vergara, and Mrs. Ma. Regina E. Estuar, for their worthwhile comments.
- My style reader, Mrs. Ma. Regina E. Estuar for her worthy assistance.
- All my respondents from Rizal Library and DISCS of ADMU for their invaluable help.
- Ate Malee, Ate Mel, and Nanette: your assistance have been so helpful for me.
- All my Philippines and Indonesian friends in Manila, especially Sr. Marivic, Joel, Rina, Erna, Bayu, Heru & family, Lukas, Fr. Heli, Fr. Guido and many others also deserve my heartfelt thanks.
- My best friend, Iskandar: thanks for being so understanding and encouraging especially when I nearly gave up. Our friendship has truly achieved.
- My sincerest gratitude goes to my family back home: my parents—Bapak & Ibu Her Suharta—, and my brother and sister—Krishna & Vera—, for their unending love, supports and prayers from afar.
- Bapak & Ibu Loekito, Ana & family, Bimo & family, Inung, & Yuni for embracing me in their family through their continuous supports and prayers.
- My beloved Mas Luki : your love always be my inspiration. Thanks for always loving me, just the way I am.
- Lastly, to the Loving God for always giving me the best in my life in very unique ways.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background of the Study

Developing a good software requires a good method. In order to fulfill this need, a number of software development methods were introduced by the mid to late 1970s. Some examples include functional decomposition approaches which are also called structured approaches, data driven or data structured approaches, and formal (mathematical) approaches (Berard 1993, 4).

According to Berard (1993, 35), the most commonly mentioned among them were the so-called structured approaches that are also called traditional or conventional approaches. Structured approaches build upon structured programming (Booch 1994, 27) that uses subprograms as basic physical building blocks. Martin (1995, 6) describes that these approaches put a great deal of emphasis on separating the definition of the data from the definition of processes. They model a software as a set of processes with data passing among them.

Despite the success of the structured approaches, the complexity of software development problems continued to grow until eventually it reached a state beyond the ability of the structured approaches to handle. Simula, which was introduced in 1966 (Berard 1993, 3) and is believed as an immediate precursor to object-oriented programming

languages, indeed provided a more prominent approach to software development. However, until the early 1980s, the presence of object-oriented programming languages merely gave impacts on the coding activity. During that time, according to Berard (1993, 25), many people still believed that the various life-cycle phases, such as analysis, design, and coding, were largely independent. For example, one could consider using structured analysis and design with object-oriented coding. Only in 1983 when Grady Booch introduced the phrase "object oriented design" did people realize that it was largely inaccurate to consider the various life cycle phases independently. Once software developers consider object-oriented approaches to software development, as Berard (1993, 25) emphasized, "it is better to have an overall object-oriented approaches".

Even though the principles underlying object-oriented methods had been foreshadowed in a number of ways since 1983, the primary focus of the object-oriented community, until 1988, was still programming language issues (Rumbaugh 1997, 16). The first object-oriented method book by Shlaer and Mellor was published in 1988. Many other method books followed quickly.

Since then people have witnessed the emergence of several object-oriented analysis and design methods. Numbers of books, articles, and papers on object-oriented analysis and design have been published in various forms. Several large projects have adopted object-oriented technology. More and more organizations have been convinced to embrace this new paradigm upon observing the positive results of these projects. The overwhelming popularity of object-oriented analysis and design method in recent years has also influenced the education field as object-oriented analysis and design became a regular course offering.

Following the rise of object-oriented analysis and design method, such "method wars" happened during 1990s in which each method claimed its superiority compared to others. Considering this kind of situation which only brought confusion to users who were in the beginning of adopting object-oriented technology, in the early 1996 the Object Management Group (OMG)[1] issued a Request for Proposal for object-oriented analysis and design interoperability. In response to the request, six proposals were submitted to the OMG in January 1997. One of them was the Unified Modeling Language (UML) 1.0. The UML was developed based on three popular methods that were introduced by Booch, Rumbaugh, and Jacobson, with contribution from several other methods. Since then, Booch, Rumbaugh, and Jacobson have been working with other proposers to reconcile their differences and to ensure support for a wide range of modeling needs into UML. In September 1997, the revised version of the UML was released as UML 1.1.

OMG finally approved UML as a standard for object-oriented analysis and design interoperability in November 1997. In addition, several Computer Aided Software Engineering tool vendors have already committed to support the UML. Many well-known companies such as Digital Equipment, Hewlett-Packard, IBM, Microsoft, and some others endorse the UML[2].

Despite all the acceptance and endorsement to the UML, another method called Business Object Notation (BON) strongly challenged the UML standardization. In an

---

[1] OMG is a non-profit corporation that was formed to create a component-based software marketplace by hastening the introduction of standardized object software. Further information about OMG is available at http://www.omg.org.

[2] http://stud2.tuwien.ac.at/~e8726711/ummw2#MU3

interview that was published in *Software Development* magazine Meyer–the designer of Eiffel language and a corresponding Computer Aided Software Engineering tool called EiffelCase that supports the BON--, claims that "compared to the UML, the BON is smaller, more powerful, and simpler" (Meyer 1997b, 54). In the same article, Meyer also explains that the BON "provides us with a method to understand the problem and to build a quality solution fast". With regard to the UML, Meyer states that the UML is too complex and not consistent enough. Furthermore, he explains:

> The idea of an analysis method is to help analyst and developers think. If they have to start reading hundred of pages of documents (such as the ones describing the Unified Modeling Language), they will never get to the real job. In contrast, the set of Business Object Notation diagrams fits on one page (Meyer 1997b, 54).

1.2 Statement of the Problem

In this growing phase of the object-oriented analysis and design methods, organizations that are planning to adopt object-oriented technology as well as educational institutions that are offering courses in object-oriented technology face one crucial question: *which method should be adopted?* Therefore, there is a need to make a practical and theoretical comparative study of the two prevailing object-oriented analysis and design methods.

1.3 Objectives

The general objective of this study is to compare two prevailing object-oriented analysis and design methods, namely Unified Modeling Language and Business Object Notation. The study is finally expected to contribute some valuable information concerning UML and BON that may help organizations and educational institutions to decide which method should be embraced. Specifically, the objectives of this study are:

- to identify the characteristics of UML and BON by performing a theoretical study on both of them;
- to implement UML and BON to a certain case study;
- to determine the comparison criteria;
- to recognize respondents' perceptions on the modeling language of UML and BON as well by conducting an informal survey among several respondents using questionnaires that are constructed based on the case study and comparison criteria;
- to compare UML and BON based on the theoretical study as well as the survey;
- to come up with conclusions and recommendations.

1.4 Scope and Limitations

The study only focuses on the UML and the BON. UML is the emerging standard for object-oriented analysis and design interoperability, while BON is one of its main opponents. Other existing methods are, therefore, beyond this study.

This study primarily focuses on the modeling language of UML and BON. Nevertheless, the process will also be discussed when relevant. As discussed in the definition of key terms, a method consists of both a modeling language and a process. During the development, software developers usually comply with the modeling language of their chosen method. However, developers may strictly follow the process advised by the chosen method, customize the advised process or even establish their own process. The decision regarding what process to follow usually depends on several factors such as organization, culture, problem domain, etc. Fowler, an expert in design and modeling with more than ten years experiences, even found that "most people, when they say they are using a method, use the modeling language, but rarely follow the process" (Fowler 1997, 1).

In other words, without lessening the importance of a process, in many ways the modeling language is the most essential part of a method for it is certainly a key of communication among the members of development team, as well as between developers and both domain experts and end-users.

It is also important to notice several limitations that are applied to the perception survey as part of this study. Due to the subjectivity that can not be avoided in the theoretical comparison, the author of this thesis feel the need of second opinions regarding the methods. This is achieved through an informal perception survey whose results in turn show the general trends of users perceptions with regard to the modeling language of the method. The results of the survey then serve as supporting data for the theoretical comparison.

The survey is *informal* and *qualitative* in a sense that it does not take statistical aspects into consideration. It is *not a comprehensive survey* for it covers only limited aspects of the method, that is the modeling language. The underlying concepts, the process as well as the techniques are not incorporated in the survey.

The questionnaires that are used for the survey are constructed based on the case study, Book Circulation System, which is chosen concerning the familiarity of potential respondents (i.e. Computer Science/Management Information System students) with the case. However, the case study can not be categorized as a large and complex system. Therefore, the results of the survey might be biased towards small to medium scale of systems.

The study ultimately generates several outputs as follows:

1. A case study along with two sets of analysis and design models. One model is built based on the UML, whereas the other is based on the BON.
2. Comparison criteria.
3. A set of tables containing the features that can only be found either in UML or BON as well as their common features in terms of concepts, modeling language and techniques, process, and environment.
4. Two sets of questionnaires to find out respondents' perceptions on the modeling language of UML and BON as well in terms of understandability, usability, and effectiveness. The first set is for system analysts/designers and programmers, whereas the second set is for domain experts/end-users.
5. Results of the perception survey that is done using the questionnaire.

6. Comparison of UML and BON based on the theoretical study as well as the survey, covering the following aspects:
    - Object-oriented approaches
    - Understandability, usability, and effectiveness of the modeling language
    - Process
    - Environment
7. Conclusions and recommendations.

1.5 Definition of Key Terms

The following are definitions of key terms that are used in this study. Despite the fact that these terms may be defined differently in other contexts, the following definitions will be consistently referred throughout the study.

Analysis is, according to Lorenz (1993, 11), the portion of the software development effort that focuses on the problem domain. The primary product of this effort is a clear, complete, verified, well-understood statement of the system requirements.

Design is the portion of software development effort that focuses on the solution domain. The primary product of this effort is a software system that accurately models the portion of the business being automated (Lorenz 1993, 11).

Method is an approach to some software engineering process (Berard 1993, 32). According to Fowler (1997, 1), most software development methods consist, at least in principle, of both a modeling language and a process. Some books use the term

methodology as a synonym for method. To be consistent, the term method will be used throughout this study.

Model, according to Webster's New World Dictionary (Agnes 1996), is a small representation of a planned or existing object.

Modeling language is the (mainly graphical) notation that methods use to express analysis and design model (Fowler 1997, 1).

Object-oriented is a term signifying that a certain system or method includes support for abstract data types combined with inheritance, polymorphism, and dynamic binding (Waldén and Nerson 1995, 399).

Object-oriented software development, according to Waldén and Nerson (1995, 400), is the process of developing software using object-oriented analysis, design, and implementation.

Process is defined by Fowler (1997, 1) as the methods' advice on what steps to take in doing analysis and design.

Software engineering, based on Que's Computer Dictionary (Pfaffenber 1992, 560), is an applied science devoted to improving and optimizing the production of computer software.

System development life cycle (SDLC) is a systematic and orderly approach to solving business problems, and developing and supporting resulting information systems (Whitten, Bentley, and Barlow 1994, 11).

CHAPTER II

REVIEW OF RELATED WORK

The history of the object-oriented paradigm can be traced from the introduction of Simula in 1966. According to Berard (1993, 3), the developers of Simula, Kristen Nygaard and Ole-Johan Dahl, introduced a number of object-oriented concepts in Simula, such as encapsulation at a higher level than mere subprograms, and the "virtual" concepts that refer to the concepts of localizing state information and operations for an object within the object itself. Furthermore Berard explains that the term "object-oriented" itself came into significant use in 1972 when Alan Kay used the term to describe the thinking behind Smalltalk which was believed as the first and archetypal object-oriented programming language (Berard 1993, 4).

After the introduction of Smalltalk, during 1980s until 1990s several other object-oriented programming languages such as Objective-C, C++, Self, Eiffel, Flavors, Trellis/Owl, and Java followed to appear.

On another hand, the issue of software development methods arose around 1970s after people realized the need to develop qualified software by avoiding bad designs from the start, and recognizing potential problem areas before software got to the production stage. Prior to the booming of software development methods in 1970s-1980s, several articles about software engineering were published during 1960s-to early-1970s. The term "software engineering" itself was selected for a 1968 NATO conference, whose purpose was to help determine what software engineering was and what it entailed (Berard 1993, 34).

By the beginning of 1980s, there were so many software development methods available, such as the top-down structured design approach driven by functional decomposition, the data structure approach, and the information hiding approach (Rumbaugh 1997, 16). These methods were covered in several software engineering books that were written by Constantine, De Marco, Yourdon, Jackson, and others.

Thus, during that time, computer community witnessed the spreading of two different popular issues: object-oriented programming languages and software development methods. In 1983 when Grady Booch introduced the phrase "object-oriented design" in his book entitled *Software Engineering with Ada* people began to realize the importance of techniques and methods to help people do good analysis and design in object-oriented software development.

As mentioned in Chapter I of this thesis, the principles underlying object-oriented method remained only as implied issues because the primary focus of the object-oriented community was still programming language issues for several years after Booch conceived the phrase. The first book describing the concepts of object-oriented method in a more explicit way is *Object Oriented Analysis and Design* by Sally Shlaer and Steve Mellor. As reviewed by Rumbaugh (1997, 16), this book contains several concepts such as classes with attributes, associations, and generalizations. Albeit the doubt that the book is not fully object-oriented, it provides a good start since a number of other object-oriented method books were quickly published after that. Some of the key books about object-oriented analysis and design methods published in the following years are briefly elaborated in the following:

1990 Wirfs-Brook, Wilkerson, and Wiener introduced Class-Responsibility-Collaboration (CRC) cards in their book entitled *Designing Object-Oriented Software.*

1990 Coad and Yourdon published a book entitled *Object Oriented Analysis* that includes classes with attributes and operations.

1990 Booch published *Object-Oriented Design with Applications* containing a discussion of classification in general as well as a number of different kinds of diagrams. It also discusses a full life-cycle process for developing systems.

1990 Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen published *Object Oriented Modeling and Design* describing OMT, a range of diagrams, and a full-cycle process.

According to Rumbaugh (1997, 17), by the end of 1990s the "method wars" were underway with readers picking their favorites. Still, many other object-oriented methods were released over the following years; some attracted notice, some not. Among them were the second generation of object-oriented methods that appeared as the original authors, as well as many others, extended and combined the first methods by preserving successful ideas and replacing weaker ideas with new ones (Rumbaugh 1997, 17). To provide a brief review of the methods that appeared after 1990, following are some examples:

1991 Peter Coad wrote a series of books on analysis, design, and programming within the object-oriented paradigm. Together with Ed Yourdon, Coad also developed Coad's lightweight and prototype-oriented approach to methods.

1991 James Rumbaugh led a team at the research labs at General Electric that produced a book about a method called the Object Modeling Technique (OMT).

1992 Object-Oriented System Engineering (OOSE) method was introduced by Ivar Jacobson.

1992 Jean Marc-Nerson introduced a method called Better Object Notation (BON) in an article on *Communication of the ACM.*

1992 Shlaer and Mellor extended their method by including dynamic modeling in Object Lifecycles.

1992 Martin and Odell published a general book presenting an object-oriented method entitled *Object-Oriented Analysis and Design,* and another book focusing on theoretical bases entitled *Object-Oriented Methods: A Foundation.*

1993 FUSION method was presented as a deliberate hybrid of OMT, Booch, CRC, and Objectory.

1993 With inputs from OMT and many other methods, Booch revised his book, and renamed it as *Object-Oriented Analysis and Design* to indicate a greater focus on the domain modeling and analysis.

1995 OMT evolved by adopting ideas from many sources, including Booch, Jacobson, Wirfs-Brock, Embley, Kurtz and Woodfield, Coleman, Martin, Odell, Shlaer and Mellor, Coad, the "Gang of Four", Selic, Gullekson and Ward, and many others.

1995 Kim Waldén joined Jean-Marc Nerson in the effort to improve the BON. A

book entitled *Seamless Object-Oriented Software Architecture* was published, based on the concept of Design by Contract firstly introduced by Bertrand Meyer. The method was renamed as Business Object Notation.

Accompanying the active period for object-oriented method ideas, some related issues in software development, such as real time systems, distributed systems, complex concurrent behavior, and modeling more complex forms of control, emerged around 1994-1995. Higher level issues of object-oriented project management were then addressed. During 1994 people also witnessed the emergence of Pattern Movement which tries to capture best practices in software development as patterns.[1]

Because of the abundance of methods available along with the contradictory claims of their superiority, the year of 1990s, indeed, were confusing for users. This fact then encouraged practitioners and researchers to do some researches and comparative studies on object-oriented analysis and design methods. The following are some of the researches that are elaborated in the Internet.[2]

- Arnold and his colleagues made a research in Hewlett Packard Laboratories to define several comparing criteria and performed an extensive comparison of five methods. The result of this research was reported in (Arnold et al. 1991, 2).

---

[1]Patterns became very popular after the publication of *Design Patterns* written by Gamma, Helm, Johnson, and Vlissides—known as Gang of Four, in 1994.

[2]http://www.cyberdine-object-sys.com/oofaq2/basic.html

- Dennis de Champeaux and Penelope Faure performed a systematic comparison of object-oriented analysis and design methods. By surveying more than ten OOADMs, they compared the common features and the major differences of the chosen methods. The result was written in (De Champeaux and Faure 1992).
- Fichman and Kemerer carried out a comparison and critique on object-oriented and conventional analysis and design methods. Two articles regarding the study written by were published in the Computer Magazine and IEEE-Computer (Fichman and Kemerer 1992a, 1992b).
- Monarchi and Puhr also performed a research topology for object-oriented analysis and design. The complete result of this research can be found in (Monarchi and Puhr 1992).
- Another comparison of ten object-oriented analysis and design methods using a formal approach was performed by Hong, Goor and Brinkkemper. They presented their result in the 26th International Conferences on System Sciences. The paper is available in the proceedings of the conference, as well as in the Internet[3].
- Wirfs-Brock and Johnson (1990) accomplished a survey on research in object-oriented design. The result of their survey was published in *Communication of the ACM* .
- In 1994 an article describing and comparing object-oriented analysis and design methods written by Fowler (1994) was also published.

[3] http://www.is.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html

Aside from the studies that are mentioned above, in order to end the "method wars" which only perplexed users, there have been several attempts over the years to bring various methodologists together to reach a consensus on core concepts, but these attempts did not succeed (Rumbaugh 1997, 18). Despite the failing of these efforts Grady Booch and Jim Rumbaugh began to work together in Rational Software Corporation to unify Booch and OMT methods in 1994. Their unification effort was not easy. According to Rumbaugh (1997, 18), finding technical agreement in semantic concepts as well as letting go of notational differences were some of the most difficult problems encountered. However, they were able to propose an initial Unified Method in October 1995. As Ivar Jacobson joined in Rational Software Corporation and in the unification effort, the incorporation of his idea on use cases then led to the newly renamed Unified Modeling Language.

Another unification effort was also performed by Henderson-Sellers, Graham, Odell, and Firesmith, based on their own methods. They came up with a unified method called OPEN in 1996. Unfortunately, according to Rumbaugh (1997, 18), this method ended unsuccessfully for not achieving widespread usage.

As described in the Background of the Study, even though in November 1997 UML has been approved by OMG as a standard for object-oriented analysis and design interoperability, there exist one challenge comes from Bertrand Meyer who believes that the BON is simpler and more powerful compared to the UML. So far there is no research that has been performed to really find which method is better between these two. By examining the two methods, this study, therefore, is intended to contribute values to the field of software engineering particularly.

CHAPTER III

METHODOLOGY

To compare the UML and the BON, this study uses the following steps:

1. Study on the UML:
   - This step focuses on the characteristics of the UML, the underlying concepts, the modeling language, the related process, and other relevant features of the UML. Different kinds of resources such as journals, books, papers, as well as information from the Internet serve as data during this step.

2. Study on the BON:
   - Similar study that is applied to the UML is also applied to the BON as well.

3. Define the comparison criteria:
   - Such criteria for comparison are defined based on the theoretical background of software engineering generally, and object-oriented analysis and design particularly.

4. Identify and classify all features of UML and BON:
   - Based on the step 1 and 2, all features of UML and BON are identified and then classified in terms of the well-defined criteria. The results of this step are presented in several tables.

5. Find common features of UML and BON:
   - All features that have been identified in the previous step are examined further during this step to find out the common features of UML and BON. Several other tables containing similarities between the two represent the results of this step.

6. Perform an informal micro-level perception survey to recognize respondents' perceptions with regard to the modeling language of UML and BON by:
   - Choose a case study, that is, **Book Circulation System**.
   - Perform analysis and design on the chosen case study by applying UML and BON as well. The results of analysis and design are two sets of models; one model is built based on the UML, whereas the other is based on the BON.
   - Prepare two sets of questionnaires based on the case study as well as the criteria. The first set is for system analysts/designers and programmers (Appendix L), while the second set is for domain experts/end-users (Appendix M). These sets of questionnaires contain the following:
     - Brief descriptions of charts and diagrams of UML and BON that will be assessed
     - Several samples of charts and diagrams that are taken from the case study
     - Questions to evaluate the charts and diagrams in terms of understandability, usability, and effectiveness
     - Questions on respondents' preferences in the case of similar diagrams that can be found in UML and BON as well.

- Distribute the questionnaires to some respondents who are taken randomly from three different groups: system analysts/designers, programmers, and domain experts/end-users. Computer Science students of Ateneo de Manila University represent the first two groups by assuming that they have enough literacy to be so. Domain experts/end-users, on the other hand are represented by some staff of Rizal Library, Ateneo de Manila University.
- Gather the questionnaires after giving the respondents enough time to fill the questionnaires out. The time that are needed by the respondents to fill them out ranges from 3 to 7 days.
- Evaluate the result of the questionnaires.

7. Evaluate UML and BON based on the theoretical study and the survey as well:
   - The well-defined criteria that have been established in the previous step are then applied to the results of the theoretical study as well as the survey on UML and BON.

8. Draw conclusions and come out with recommendations:
   - The last step of this research is to come out with conclusions regarding the comparison of the UML and the BON. Recommendations then are formulated to assist organizations and educational institutions that intend to adopt object-oriented analysis and design.

CHAPTER IV

THEORETICAL BACKGROUND

4.1 Basic Object-Oriented Concepts

4.1.2 Object

Object technology builds upon object model. An *object* is a software "packet" containing a collection of related data elements and a set of procedures called methods, for operating on these elements (Taylor 1990, 12). Thus, an object is composed of two important things: data and methods.

4.1.2 Encapsulation

As stated above, the object packages the data and its methods together. This arrangement is called *encapsulation*. Encapsulation hides the details of its internal implementation from the users of an object (Martin and Odell 1994, 17). By allowing the data to be accessed only via its own methods, encapsulation also hides the data from other objects and therefore, protects the data from corruption.

4.1.3 Message

An object oriented application can be stated as a set of objects working together to support a common purpose (Martin 1995, 13). To be able to collaborate, objects should communicate with one another through messages. A *message* is a request to carry out the indicated method on a given object and return the result (Martin and Odell 1994, 19). Thus, to make an object do something, a message that constitutes a request contains the name of the object, the name of the method to be invoked, and sometimes a group of parameters should be sent to it.

When an object A sends a message to object B, a method of object A invokes a method of object B resulting in the data of object B being manipulated in some way. Object B may then return a value to object A.

4.1.4 Class

A set of objects that share a common structure and a common behavior can be categorized into a *class* (Booch 1994, 103). A single object is simply an instance of a class. For example, `Borrower` applies to the objects who are persons borrowing a book in a library. Instances of `Borrower` could be Jonathan Meyer, Mary Graham, and so on. In this example, `Borrower` is a class while Jonathan and Mary are objects.

A class specifies a data structure and the permissible operational methods that apply to each of its objects. For example, a `Borrower` class may include data about ID number,

name, status, address, etc. A method associated with the object type `Borrower` might be one that accepts a new borrower, or changes the borrower status.

4.1.5 Inheritance

A high-level class can be specified into lower-level classes. A lower-level class inherits properties of its parent class. This arrangement is called *inheritance*, which is defined by Booch (1994, 112) as a relationship among classes wherein one class shares the structure and/or behavior defined in one (*single inheritance*) or more (*multiple inheritance*) other classes.

The class from which another class inherits is called *superclass*, whereas the class that inherits from one or more classes is called *subclass*. As well as inheriting the data structure and methods, or some of the methods, of its superclass, a subclass may also have methods and sometimes data types of its own. For example, figure 4.1 shows a class and its subclass. The subclass inherits the methods of its superclass but also has its own method, D.

4.1.6 Polymorphism

Objects provide the dynamic behavior that will be performed when these objects start to communicate with each other. An object may know of other objects to which messages (i.e. requests for an operation) can be sent. If an object sends a message to another object, but does not have to be aware of which class the object belongs to, it is called *polymorphism* (Jacobson 1992, 55).

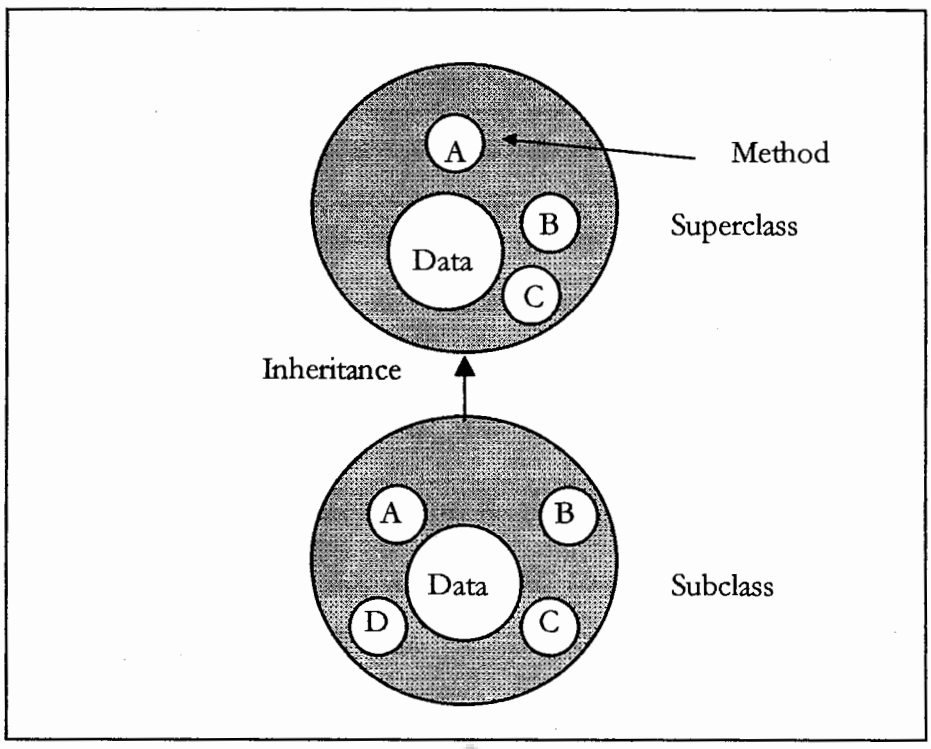


Figure 4.1 A superclass and its subclass
(Source: Martin and Odell 1994, 23)

One strength of polymorphism is that a request for an operation can be made without knowing which method should be invoked (Martin and Odell 1994, 113). These implementation details are hidden from the users. Instead, the selection mechanism of the object-oriented implementation takes the responsibilities.

4.2 Object-Oriented Software Development

The process of developing object-oriented software includes object-oriented analysis, design, and implementation. Object-oriented analysis focuses the development effort on the

problem domain. As mentioned in the Definition of Key Terms, the primary product of this effort is a clear, complete, verified, well-understood statement of the system requirements.

During the design, the focus of the development shifts to the solution domain. A software design model that accurately represents the portion of the system being automated is the primary product of this effort.

An object-oriented software design model can be classified into two different descriptions: *static model* and *dynamic model*. Static model describes the structure of a system: what the components (i.e. classes) are and how these components are related to each other (Waldén and Nerson 1995, 24). Dynamic model, according to Waldén and Nerson, document how the system will behave over time. In an object-oriented context, this means how objects interact at execution time; how they invoke operations on each other; and how the information content of the system changes, as reflected by the values of the class attributes (state variables) in the system (Waldén and Nerson 1995, 24).

As the software development effort proceeds to the implementation stage, the design model is then implemented using any programming language to finally produce the real software system for the specified problem domain.

4.3 The Unified Modeling Language

Booch, Rumbaugh, Jacobson and a team that consists of representatives of several companies worldwide developed the Unified Modeling Language. As described in Chapter II, the development of UML began as an effort to overcome "method wars" which was

undergoing during 1980s, by providing such a unified method. Through iterative and incremental process over the years, UML reaches its current state.

Several goals were established during the design of UML[1] . Following are these goals:

1. Provide users a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the object-oriented tools market.
6. Support high-level development concepts such as collaborations, frameworks, patterns, and components.
7. Integrate best practices.

As described in UML homepage, UML is defined as "a language for specifying, visualizing, constructing, and documenting the artifacts of software systems". In order to provide users with a comprehensive understanding, UML provides the following documents:[2]

1. *UML Semantics* defines the semantics and syntax of the UML using three consistent views: abstract syntax, well-formedness rules, and semantics.
2. *UML Notation Guide* defines notation representing the graphic syntax for expressing the semantics and provides supporting examples as well. Several standard diagram types, as

---

[1] http://www.rational.com/uml/summary.html

[2]The complete documents are available at http://www.rational.com/uml

listed in Appendix A, are defined in the UML Notation Guide to enable users depicting a system at different levels of views.

3. *UML Extensions* provide users with facilities so that they can use UML in process-and domain-specific developments. Two extensions are currently available, namely *Objectory Process for Software Engineering* –which has been revised into a newer version called *Rational Unified Process*–, and *Business Modeling*.

The developers of UML declare that UML focuses on modeling language, not on process because they believe that although UML must be applied in the context of a process, the experiences have shown that different organizations and problem domains require different processes. However, UML still gives attention to process for software engineering by providing extension documents called the Objectory Process and its newest version called Rational Unified Process.

By defining three kinds of documents that are mentioned above, UML promotes a development process that is u*se-case driven, architecture centric,* and i*terative and incremental*

4.3.1 Unified Modeling Language Diagrams

To exhibit the analysis and design models, UML provides the following graphical diagrams:

1. Use case diagram
2. Class diagram

3. Behavior diagrams:
    a. Statechart diagram
    b. Activity diagram
    c. Interaction diagrams:
        - Sequence diagram
        - Collaborations diagram
4. Implementation diagrams:
    a. Component diagram
    b. Deployment diagram

These diagrams provide multiple perspectives of the system under analysis or design. The underlying model integrates these perspectives so that a self-consistent system can be analyzed and built. More detail descriptions of these diagrams can be found in Appendix A.

4.3.2 Rational Unified Process

Rational Unified Process is a web-enabled, searchable knowledge base that enhances team productivity and delivers software best practices through guidelines, templates, and tool mentors for all software development activities[3]. Rational Unified Process actually is not a core part of the UML. Yet, it is an extension of the UML that has been developed hand-in-hand with the UML, allowing developers to gain the full benefits of UML. For the purpose of this study, Rational Unified Process is discussed in compared to BON process.

---

[3] http://www.rational.com/uml/html/notation/

Rational Unified Process can be described in two dimensions, or along two axes:

- The *horizontal axis* represents time and shows the dynamic aspect of the process as it is enacted, and it is expressed in terms of cycles, phases, iterations, and milestones.
- The *vertical axis* represents the static aspect of the process: how it is described in terms of activities, artifacts, workers, and workflows.

Further discussions on Rational Unified Process are presented in Appendix C.

4.4 The Business Object Notation

Business Object Notation (BON) is an object-oriented analysis and design method that presents a set of concepts for modeling object-oriented software, a supporting notation in two versions –one graphical and one textual– and a set of rules and guidelines to be used in producing the models (Waldén and Nerson 1995, 11). Thus, as explained further by Waldén and Nerson (1995, 120), BON provides a recommended analysis and design *process* consisting of nine major tasks to be carried out. Each task uses a set of *input sources* and produces a set of well-defined *deliverables* (either new or updated) as part of the end result.

BON was firstly introduced by Jean-Marc Nerson (1992) in *Communication of the ACM* as a graphical formalism for representing system structures (Meyer 1997a, 919). The original name was Better Object Notation. When Kim Waldén co-authored with Jean-Marc Nerson (1995) published *Seamless Object-Oriented Software Architecture*, the notation was then developed from just a notation to a complete development method that includes basic underlying

concepts, notation, process, and standard activities. The name was then changed into Business Object Notation.

4.4.1 Principles and Characteristics of the Business Object Notation

BON emphasizes on three principles namely *seamlessness*, *reversibility*, and *software contracting*. Seamlessness is the use of a consistent set of concepts and notations throughout the software development lifecycle.

Reversibility is the support for both forward and backward engineering: from analysis to design and implementation, and back (Meyer 1997a, 919). Reversibility guarantees that changes made at any step in the process, even as late as detailed implementation or maintenance, can be reflected all the way back to the earlier steps, including analysis, so as to guarantee the consistency of the entire project baseline[4].

Software contracting[5] is a software development principle that views the construction of a software system as a succession of precise contracts between its classes, to guarantee reliability and consistency.

Aside from the three principles that are explained above, BON also emphasizes on several other characteristics which are briefly explained in the following.

---

[4] http://www.eiffel.com/products/bon.html

[5] "Software contracting" is a design technique introduced by Bertrand Meyer and has been appreciated as an important advance in the quest for correctness and robustness software products as well as reusability. To get more information about software contracting, see (Meyer 1997a).

The first characteristic is *generality* which means that BON concentrates on what is essential for object-oriented development in general, and tries to define a consistent notation to support the corresponding concepts (Waldén and Nerson 1995, 17). This characteristic provides users with such a flexibility to complement the notation with whatever needed for particular projects.

The second characteristic is *scalability*. Developing large systems requires such a notation that should be able to represent large structures and views at different levels of details in order to provide comprehensive views of the whole systems. BON fulfils the need of scalability by providing facilities called *nested clustering* and *element compression*.

Clustering is a facility to group classes into higher-level units. Element compression means representing a set of graphical or textual elements with a simpler element, its compressed form (Waldén and Nerson 1995, 19). BON allows users to freely choose the amount of detail shown for each part of a system, since the level of compression can be independently selected for each structural element recursively.

Typed interface description is the third characteristic of BON. In order to provide an essential aid for system specification, BON adopts a fully typed notation for class interfaces. Thus, a class interface consists of the syntax and semantics of its operations. The syntactic part of an operation, called its *signature*, consists of its name, the number and types of its arguments (if any) and the type of its return value (if any).

In addition to the fully typed notation for class interface, BON also provides a very high-level untyped view in a form of textual charts to be used in communication with non-technical people (Waldén and Nerson 1995, 10).

Another characteristic of BON is *simplicity*. Waldén and Nerson (1995, 22) emphasize that the notation strives for simplicity and tries to minimize the number of concepts.

The designers of BON believe that any notation designed should avoid wasting space without loosing ability to give a global overview of a potentially large and complex structure which is important regardless the details. Providing compressed form for all space-consuming graphical layouts then manifests one characteristic of BON, *space economy*.

According to Waldén and Nerson (1995, 24) one of the great advantages with the increased semantic content in a typed object-oriented notation with software contracts is that it provides the foundation for much more intelligent object-oriented CASE tools than is possible in untyped environments. The *basis for intelligent case tools*, therefore, is claimed as one of BON's characteristics.

4.4.2 Business Object Notation Charts and Diagrams

To exhibit static and dynamic model, BON has two variants of notation: graphical BON and textual BON (Waldén and Nerson 1995, 29). The graphical form is intended for use with automatic tool supports as well as for sketches on paper and whiteboards. The textual form is intended for communicating BON descriptions between various automatic processing tools and for maintaining evolving architectures by simple file editing in environments that lack dedicated BON CASE tools. There is one-to-one structural mapping between the graphical and textual form of a BON description. This study, however, will only focus on

the graphical notation since it is more communicative and used more often than the textual one.

Following are the static charts and diagrams that are provided by BON to exhibit static model of the system:

1. System chart
2. Cluster chart
3. Class chart
4. Static architecture
5. Formal class description

The dynamic model of the system on the other hand can be described using the following charts and diagrams:

1. Event chart
2. Scenario chart
3. Object creation chart
4. Object scenario.

Further discussions about static and dynamic model of the BON can be found in Appendix D.

4.4.3 Process and Standard Activities

BON relies on the evolving class descriptions, cluster charts, and object scenarios to keep track of visibility issues. The method defines nine major tasks for analysis and design

process. These tasks are loosely grouped in three phases with the following aim (Waldén and Nerson 1995, 150):

- Task 1 - 3: gathering analysis information.
- Task 4 - 6: describing the gathered structured.
- Task 7 - 9: designing a computational model.

Although the tasks are often listed in sequential order, Nerson (1992, 66) says that they are usually iterative in practice. Appendix H elaborates these tasks along with their corresponding general results as well as input sources, deliverables, and acceptance criteria. Since the tasks are listed in approximate order of execution, and represent the ideal process, BON users are free to change the order of the tasks or make any other deviations from the advised process that will help fulfil the goals of a particular project, as long as the required static and dynamic models are eventually produced (Waldén and Nerson 1995, 148).

The goal of BON process is to gradually build the products or deliverables shown in Appendix E.

Aside from the process, BON also defines nine standard activities that a developer will be involved with as part of performing the tasks defined by BON process. These activities are listed in Appendix H.

Waldén and Nerson (1995, 178) describe that these standard activities are *orthogonal to the process* in the sense that they may all occur to some degree as part of many of the BON process tasks. The first four are continuously repeated during both analysis and design. The fifth occurs mostly during analysis, but may also be repeated during design of large systems. The last four, finally, are chiefly part of the design tasks.

CHAPTER V

RESULTS OF THE STUDY

5.1 Comparison Criteria

According to Pfleeger (1992, 9), in order to come up with a quality software several factors need to be considered namely *principles*, *concepts*, *techniques*, and *tools*. Whereas, Quatrani (1998, 4) mentions that there are three components needed for a successful software project. She calls these components as a triangle for success, which consist of *process*, *notation*, and *tool*.

Their ideas about the key elements of software projects are complementary. Nevertheless, the author of this thesis believes that two more important elements should also be considered. These two elements are *people*, which includes technical people (system analysts, designers, and programmers) and non-technical people (domain experts, users or customers), and *programming language*.

Thus, as shown in figure 5-1, there are six key components that should be taken into account in software development projects, namely *people*, *concepts*, *modeling language and techniques*, *process*, *tools*, and *programming language*. Each component plays an important role in every software project. Lack of consideration of one component might cause the projects fail.

These key components do not stand independently. Instead, they are closely interwoven. People (i.e. developers in this case) need clear and robust underlying concepts

to analyze the problems and to come up with suitable solutions. Complete and comprehensive modeling language and techniques are important for developers to communicate problem analysis and its corresponding design among the member of project team as well as to communicate with non-technical people. However, the developers will probably fail if they do not know how to use the modeling language and techniques in the process of developing a software. Furthermore, if the developers can not document the artifacts of their work with a reliable tool, they will probably fail. Lastly, the result of the design needs to be implemented using a robust programming language that can capture all the specifications elaborated in the design.

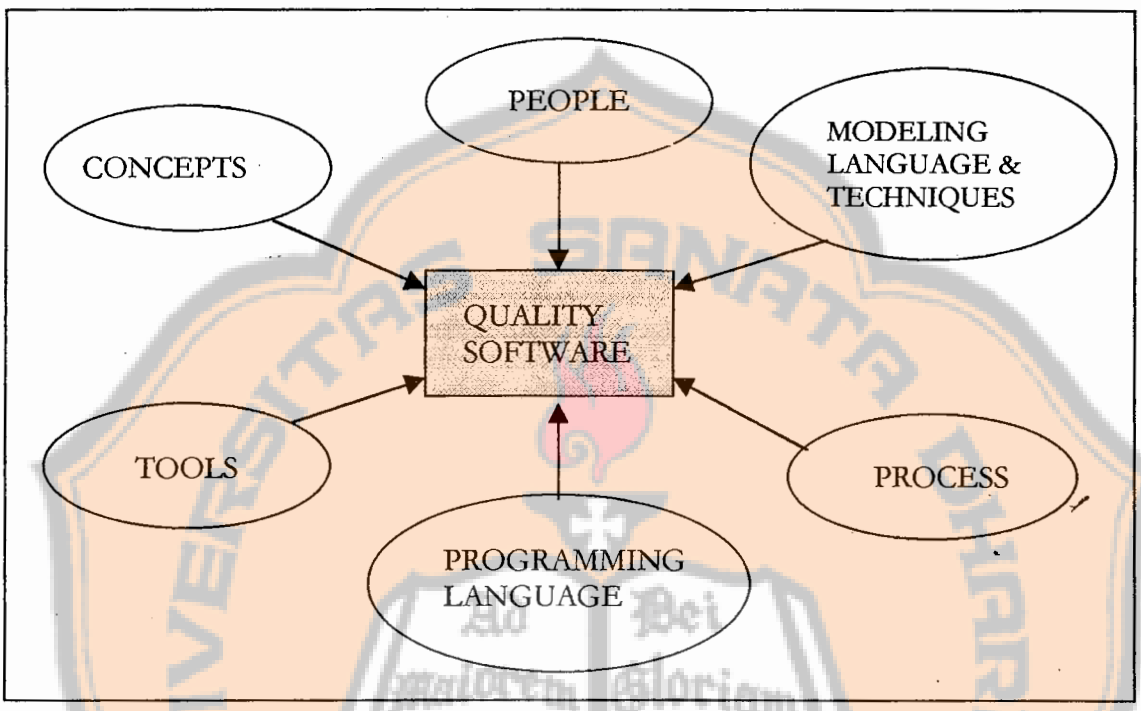


Figure 5-1 Key components of software projects

From these key components several criteria to compare the BON and the UML are then determined. Table 5-1 shows these criteria, whereas table 5-2 describes a mapping from key components of software projects shown in figure 5-1 to the comparison criteria.

Table 5-1 Comparison Criteria

NO	GENERAL CRITERIA	DETAIL CRITERIA
1.	**Object-oriented approaches:** • How does the method encourage sound object-oriented approaches?	1. Does the method provide complete underlying object-oriented concepts which cover: • Basic object-oriented concepts • Static aspects of objects • Dynamic aspects of objects 2. Does the method provide complete, comprehensive, and consistent modeling language & techniques? 3. Does the method provide any advantages in terms of object-oriented approaches?
2.	**Understandability, usability and effectiveness of the modeling language:** • Does the method provide understandable, usable, and effective modeling language for both technical and non-technical people?	1. Are its modeling languages understandable? 2. Are its modeling languages usable? 3. Are its modeling languages effective to help people in describing and understanding the problem domain as well as the solution domain? 4. In the case of common diagrams, are the diagrams preferable for its users?
3.	**Process:** • How does the method provide supports to object-oriented process?	1. Does the method advise a well-defined object-oriented software development process? 2. Is the advised process iterative? 3. Is the advised process configurable? 4. Does the advised process provide any advantages?

4.	**Environment:** • How is the environment within which the method exists?	1. Is there any existing tool that supports the method? 2. Is there any successful project that has been done using the method? 3. Does the method have supporting users? 4. Are there many available literatures on the method?

Table 5-2 Mapping of Key Components of Software Projects to Comparison Criteria

KEY COMPONENT OF SOFTWARE PROJECT	COMPARISON CRITERIA
1. Concepts 2. Modeling language & techniques 3. People	Object-oriented approaches
1. Modeling language 2. People	Understandability, usability, & effectiveness of modeling language
1. Process 2. People	Process
1. Tools 2. People	Environment

From the mapping in table 5-2, it is obvious that each criterion is derived from a combination of "people" with one or more key components of software projects. It represents a fact that people, indeed, is the center of every software development projects. It is the people who will use all the underlying concepts, the modeling languages, the techniques, and the tools. It is also the people who will implement the advised process. Therefore, to evaluate a method in every aspects of it, involving people's point of view is a must.

Table 5-2 shows no mapping from comparison criteria to the *programming language* component. This particular component, which is much closer to implementation aspect, is not taken into account in this study merely because the focus of this study is more on analysis and design aspect.

5.2 Phase of Comparison

As described in Chapter III, the comparison steps are divided into four main phases, namely identification, classification, informal perception survey, and evaluation. Table 5-3 elaborates further the purpose of each phase, along with its corresponding output.

Table 5-3 Phase of Comparison

NO	PHASE	PURPOSE	OUTPUT
1.	Identification and classification	• To identify all features of UML and BON. • To classify the features in terms of concepts, modeling language & techniques, process, and environment	1. 4 tables containing UML features on: • object-oriented concepts • modeling language and techniques • process • environment 2. 4 tables containing BON features on: • object-oriented concepts • modeling language and techniques • process • environment
2.	Finding common features	To find similarities between UML and BON	1. 4 tables containing common features on:

			• object-oriented concepts • modeling language and techniques • process • environment 2. 4 tables containing features that can only be found in UML in terms of: • object-oriented concepts • modeling language and techniques • process • environment 3. 4 tables containing features that can only be found in BON in terms of: • object-oriented concepts • modeling language and techniques • process • environment
4.	Informal Perception Survey	1. To find out respondents' perceptions regarding the modeling languages of UML and BON in terms of: • Understandability • Usability • Effectiveness 2. To find out respondents' preferences with regard to similar diagrams, which are provided by both UML and BON	1. A table containing respondents' perceptions on UML diagrams 2. A table containing respondents' perceptions on BON charts/diagrams 3. A set of tables containing respondents' preferred diagrams
5.	Evaluation	To evaluate each entry of all tables resulted from step 1 to 3, based on the detail criteria.	A set of tables containing the answers to the detail criteria.

Figure 5-2 provides an illustration on how phase 1 and 2 of comparison are performed. The circles symbolize UML and BON. The intersection of these two circles represents the common features of UML and BON.

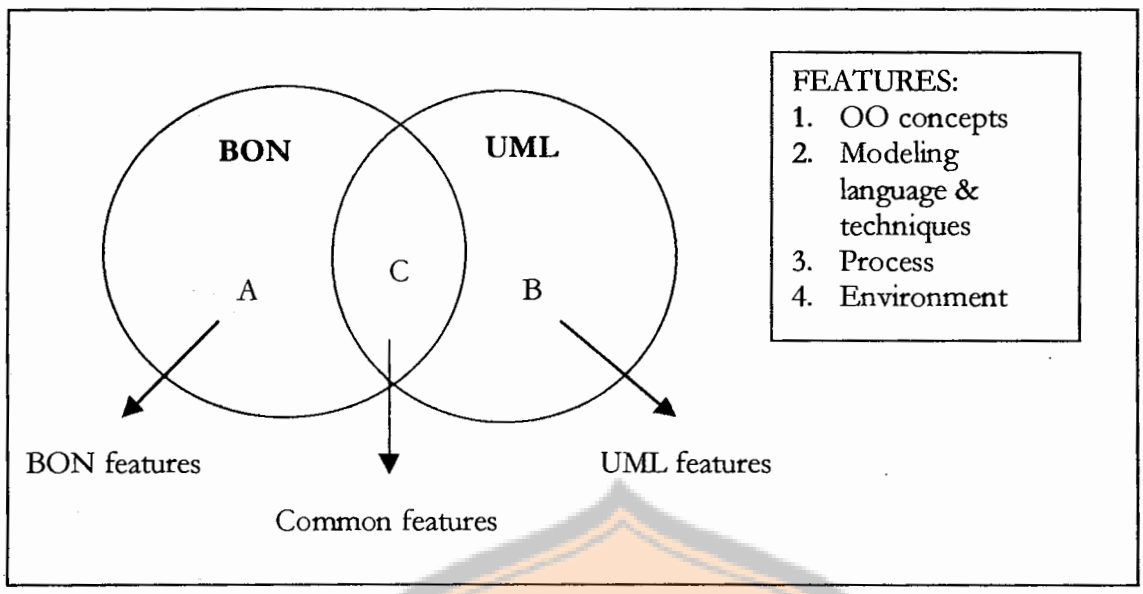


Figure 5-2 Identification, Classification & Finding Common Features

5.3 Result of the Comparison

5.3.1 Phase 1 & 2: Identification, Classification, and Finding Common Features

Extensive results of phase 1 and 2 of the comparison can be found in Appendix Q. Notice that each table of common features also contains a column describing the discrepancies that might exist between the common features. Even though a feature can be found both in UML and BON, sometimes the meaning as well as the coverage of the feature is not exactly

the same. On the other hand, in some other cases, the term that is used in both UML and BON is not the same but it represents the same meaning and coverage. Figure 5-3 to 5-6 in the following also presents summary of the results of phase 1 and 2.

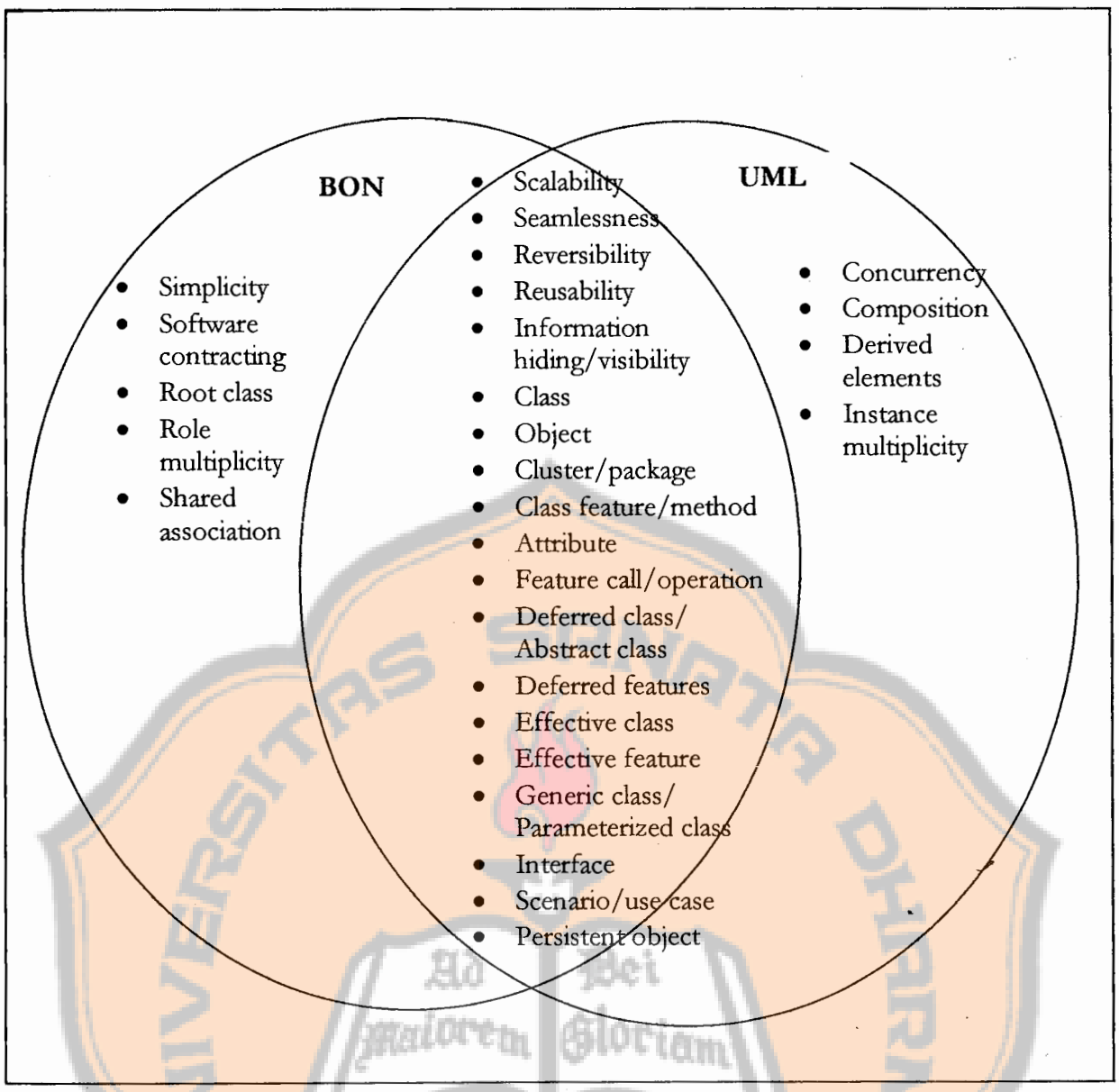


Figure 5-3 Features on object-oriented concepts

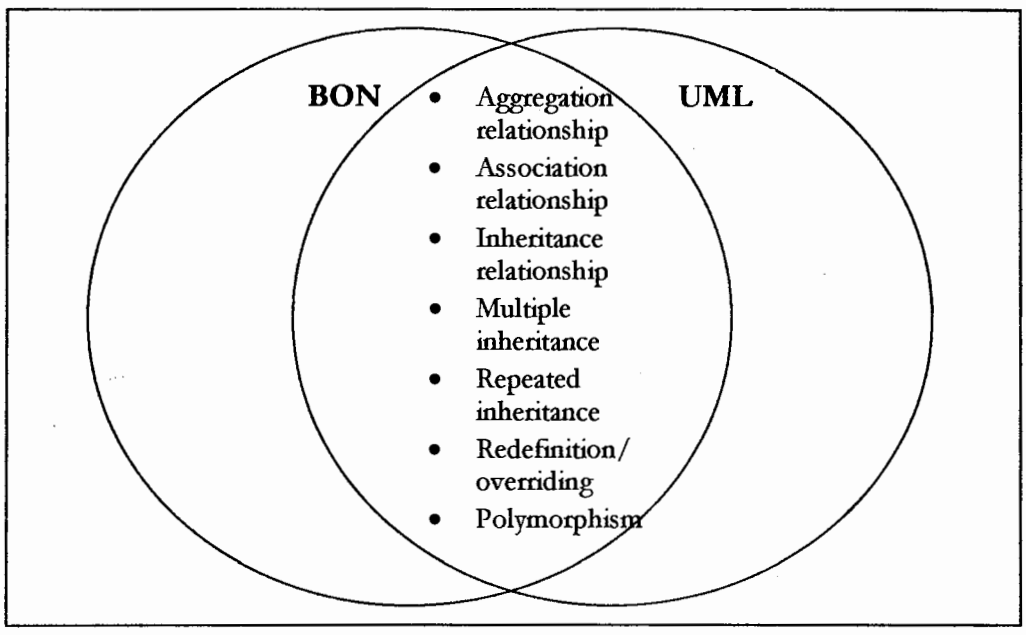


Figure 5-3 Features on object-oriented concepts (cont.)

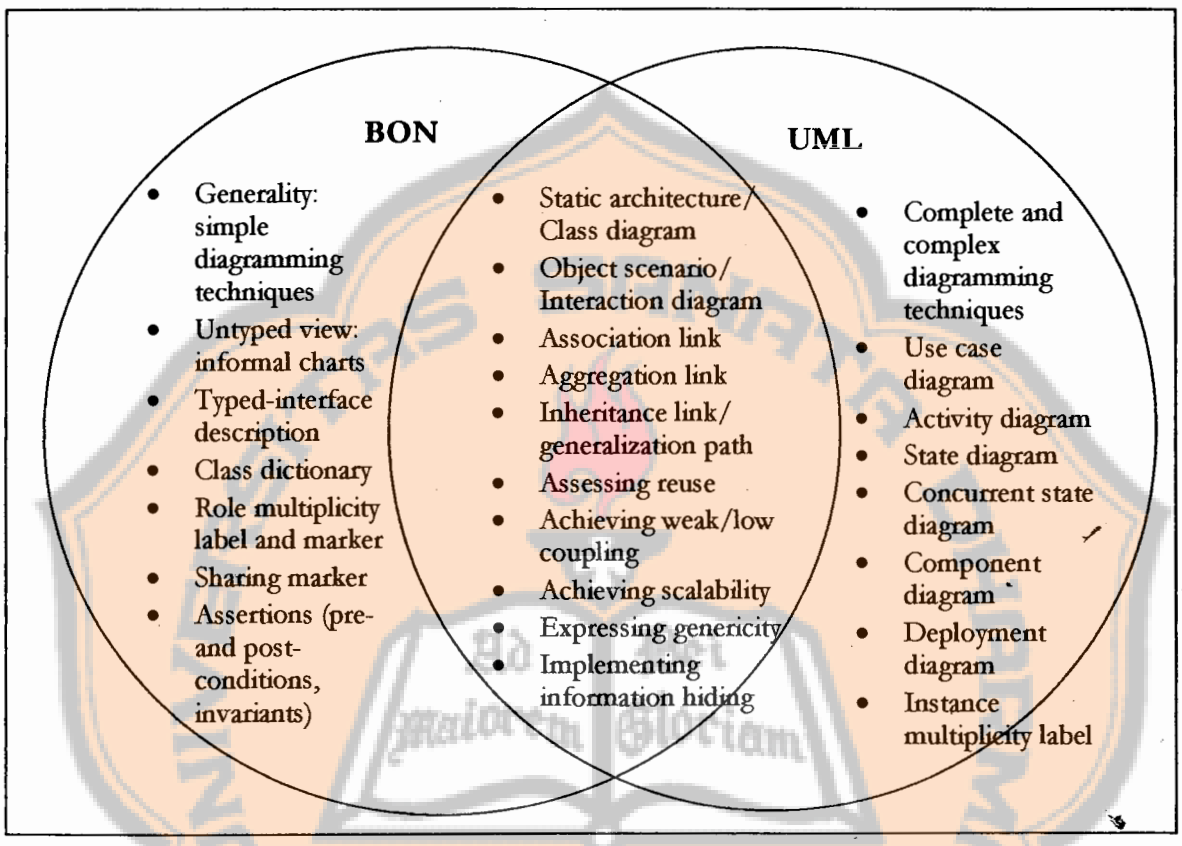


Figure 5-4 Features on modeling language and techniques

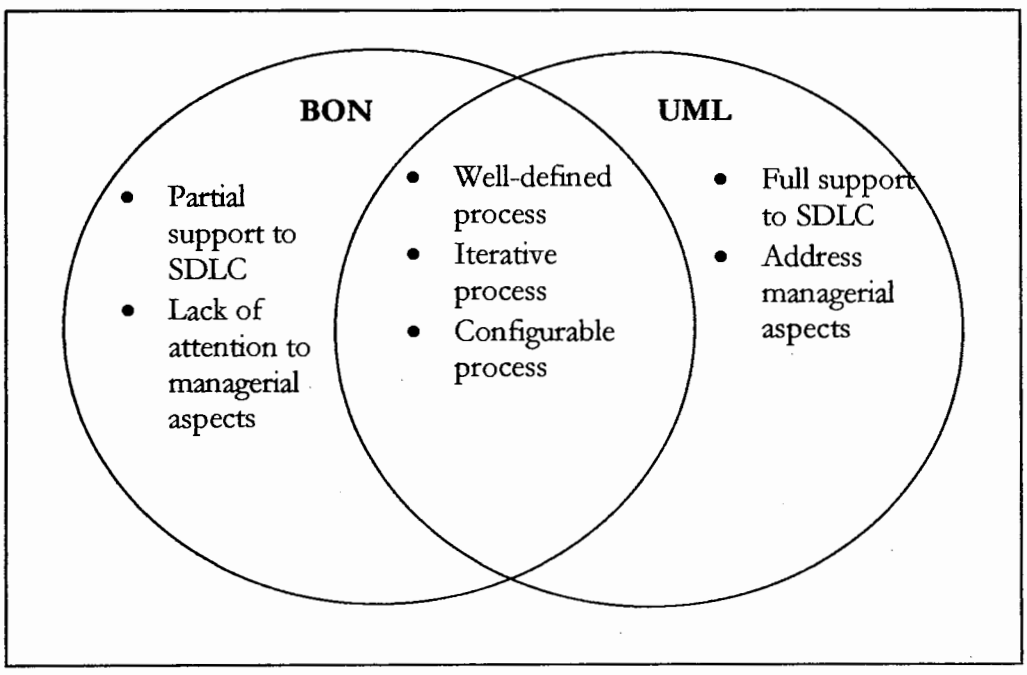


Figure 5-5 Features on Process

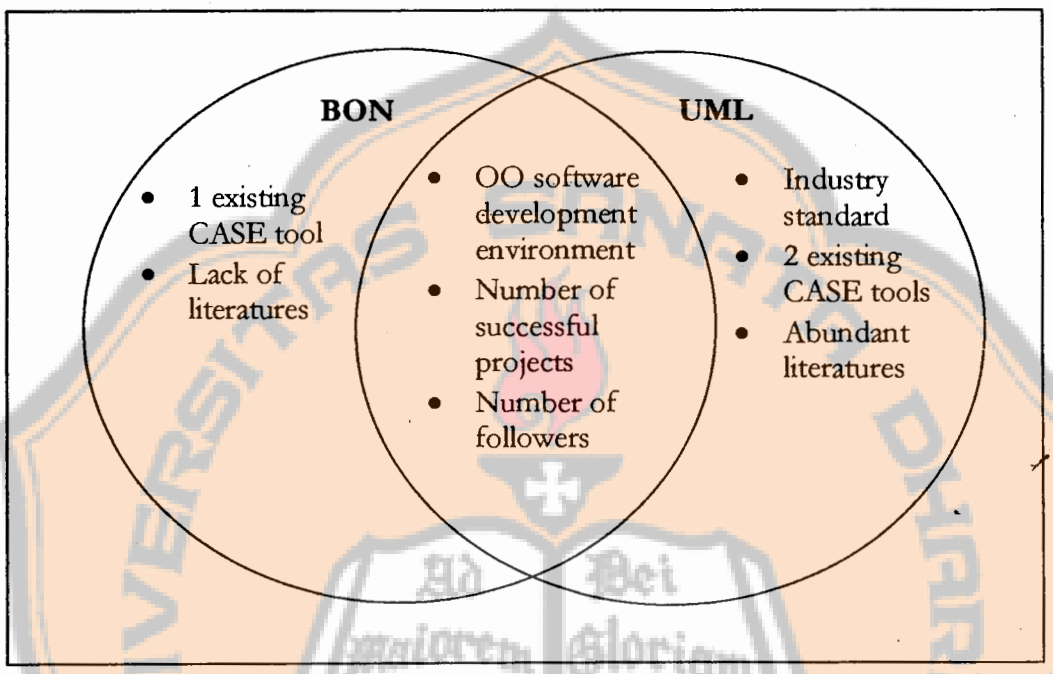


Figure 5-6 Features on Environment

5.3.2 Phase 3: Informal Perception Survey

In order to determine whether the modeling languages (i.e. charts and diagrams) of both UML and BON are understandable, usable, and effective, an informal perception survey is conducted. The main purpose of the survey is to find out respondents' perceptions on several UML diagrams as well as BON charts and diagrams in terms of understandability, usability, and effectiveness. In the case of two diagrams with functional similarity —one belongs to UML, whereas the other belongs to BON—, the survey is also meant to recognize respondents' preferences between these two.

Prior to the survey, a case study, that is **Book Circulation System**, is chosen to be analyzed and designed using UML and BON. The results of this analysis and design are two sets of model; one in UML whereas the other one in BON. Appendix J and K provide complete documentation of these two models respectively.

Two sets of questionnaires (see Appendix L and Appendix M) as instruments for the survey are then prepared to be distributed to respondents who are taken from three different groups: system analysts/designers, programmers, and domain experts/end-users. By assuming that they are capable to become system analysts/designers and programmers as well, ten Computer Science/Management Information System students of Ateneo de Manila University represent the first group whereas seven among them also represent the group of programmers at the same time. Five staffs of Rizal Library of Ateneo de Manila University, on the other hand, represent the last group.

The staffs of Rizal Library of Ateneo de Manila University who are chosen as respondents are the ones in charge either in circulation section or technical section. They are asked to evaluate three diagrams of UML namely use case diagram, class diagram, and activity diagram, as well as one chart and one diagram of BON namely cluster chart and static architecture.

Among the respondents from the group of domain experts/end-users, only two of them, the technical assistants, are familiar with programming language (i.e. Xbase). However, almost all of them (4 out of 5) posses a sufficient background in computer literacy and have experiences in computerization process. Appendix P furthermore presents their profiles.

Appendix N on the other hand presents the profile of respondents who are randomly chosen from Computer Science/Management Information System students of Ateneo de Manila University to represent groups of system analysts/designers and programmers.

Aside from respondents' understandability on object-oriented concepts, the questionnaire that is given to system analysts/designers and programmers contains 2 charts and 10 diagrams of UML and BON. Not all of charts and diagrams that are provided by UML and BON are included in the questionnaire considering that the questionnaire should be simple to be filled out and yet representative enough to evaluate the modeling language. Concerning the similarity of BON charts, only two of them are included in the questionnaire: class chart and cluster chart. Class chart is chosen for it is one of the most

important charts in BON. The other charts are quite similar with cluster chart in terms of form and function. Hence, cluster chart is chosen to represent this similar group of charts.

In the case of interaction diagrams, only one type of them—that is collaboration diagram--, is chosen to represent this kind of diagram. Sequence diagram, the other type of interaction diagram, is not included in the questionnaire.

Table 5-4 to 5-6 presents the summary of the survey's results, which is elaborated more complete in Appendix N, Appendix O, and Appendix P.

Table 5-4 Summary of the Results of Survey on BON Charts/Diagrams

CRITERIA	RESPONDENT	YES	NO
Textual descriptions are More understandable?	DEU (5)	1	4
	SAD (10)	3	7
	P (7)	4	2

Legend:

- DEU : Domain Experts/End-users (# of respondents = 5)
- SAD : System Analysts/Designers (# of respondents = 10)
- P : Programmers (# of respondents = 7)

Table 5-4 Summary of the Results of Survey on BON Charts/Diagrams (cont.)

DIAGRAM	UNDERSTANDABILITY						USABILITY						EFFECTIVENESS			
	DEU (5)		SAD (10)		P (7)		DEU (5)		SAD (10)		P (7)		SAD (10)		P (7)	
	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Cluster chart	1	4	8	2	6	1	4	1	10	0	7	0	10	0	5	2
Class chart	NA	NA	8	2	6	1	NA	NA	8	2	6	1	8	2	6	1
Static architecture	2	3	8	2	7	0	3	1	8	2	6	1	8	2	6	1
Formal class description	NA	NA	8	2	6	0	NA	NA	7	3	6	0	9	0	6	0
Object scenario	NA	NA	8	2	6	1	NA	NA	8	2	5	2	9	1	6	1

Legend:

- DEU : Domain Experts/End-users (# of respondents = 5)
- SAD : System Analysts/Designers (# of respondents = 10)
- P : Programmers (# of respondents = 7)
- Y : YES (Understandable/usable/effective)
- N : NO (NOT understandable/NOT usable/NOT effective)
- NA : Not Applicable

Table 5-5 Summary of the Results of Survey on UML Diagrams

DIAGRAM	UNDERSTANDABILITY						USABILITY						EFFECTIVENESS			
	DEU (5)		SAD (10)		P (7)		DEU (5)		SAD (10)		P (7)		SAD (10)		P (7)	
	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Use case diagram	5	0	10	0	6	1	5	0	9	1	6	1	9	1	7	0
Class diagram	4	1	8	2	6	1	3	1	9	1	7	0	9	1	7	0
Collaboration diagram	NA	NA	7	3	7	0	NA	NA	7	3	7	0	8	2	7	0
Activity diagram	5	0	10	0	6	1	5	0	10	0	6	1	10	0	7	0
State diagram	NA	NA	8	2	5	2	NA	NA	7	3	7	0	7	3	6	0
Package diagram	NA	NA	8	2	4	3	NA	NA	5	5	3	4	6	4	3	4
Deployment diagram	NA	NA	5	5	5	2	NA	NA	6	4	7	0	5	4	4	2

Legend:

- DEU : Domain Experts/End-users (# of respondents = 5)
- SAD : System Analysts/Designers (# of respondents = 10)
- P : Programmers (# of respondents = 7)
- Y : YES (Understandable/usable/effective)
- N : NO (NOT understandable/NOT usable/NOT effective)
- NA : Not Applicable

Table 5-6 Summary of the Results of Survey on Preferences

a. Class diagram vs. Static architecture and formal class descriptions

CRITERIA	RESPONDENT	STATIC ARCHI + FORMAL CLASS DESCRIPTION	CLASS DIAGRAM
More understandable	DEU (5)	0	5
	SAD (10)	5	4
	P (7)	2	5
More effective to describe static Structure of the system	SAD (5)	5	4
	P (7)	2	5
More effective to help during implementation stage	P (7)	2	5

b. Object scenario vs. collaboration diagram

CRITERIA	RESPON-DENT	OBJECT SCENARIO	COLLABORA-TION DIAGRAM	NO PREFERENCE
More understandable	SAD (10)	6	1	0
	P (7)	3	3	1
More effective to describe object collaboration	SAD (10)	5	3	0
	P (7)	0	5	2
More effective to help during implementation	P (7)	0	6	1

5.3.3 Phase 4: Evaluation

Table 5-7 to 5-10 presents the results of phase 4. Each row of the tables is related to one entry of the general criteria, as can be seen in the second column of the table. To differentiate the advantage features from the disadvantage ones, a "+" sign is assigned to each advantage whereas a "–" sign marks each disadvantage.

In the following part, the results of the study are organized and discussed extensively based on the four general criteria.

5.3.3.1 Object-oriented approaches

Table 5-7 Comparison on object-oriented approaches

NO	CRITERIA	BON	UML
1.	Complete underlying concepts?	+ Complete underlying concepts	+ Complete underlying concepts
2.	Complete, comprehensive, & consistent modeling language & techniques?	+ Comprehensive & consistent modeling language & techniques – Lack of graphical representation to describe relationships between users & scenarios – Lack of workflow diagram – Lack of state diagram	+ Comprehensive & consistent modeling language & techniques + Complete modeling language – Too complex modeling language – Gives no sufficient attention to rules and constraints (software contracting) – Lack of class dictionary
3.	Advantages?	*Characteristics:* + Scalability + Seamlessness + Reversibility + Generality + Simplicity *Modeling language & techniques:* + Software contracting + High-level untyped-view (informal charts) + Typed-interface description	*Characteristics:* + Scalability + Seamlessness + Reversibility *Modeling language & techniques:* + Use case diagram + State diagram + Activity diagram + Package diagram + Deployment diagram + Component diagram + Concurrency handling

		+ Class dictionary + Role multiplicity + Shared association	+ Composition + Derived elements + Instance multiplicity

Completeness of underlying concepts

Both UML and BON provide complete underlying object-oriented concepts. All necessary features that are needed to represent basic object-oriented concepts, static aspects of objects, as well as dynamic aspects of objects are supported by the methods. Some of the features are referred using the same term in both methods, whereas some others are referred by different terms. Despite the differences in terms, the meanings still the same.

Completeness, comprehensiveness and consistency of modeling language and techniques

UML as well as BON provide comprehensive and consistent modeling language and techniques in a sense that all components of the modeling language and techniques are interrelated to build the whole complete system. However, comprehensiveness and consistency are easier to observe in BON rather than in UML since BON strongly emphasizes on seamless and reversibility approaches that provide a uniform perspective of the system, based on class descriptions and object scenarios. UML, on the contrary, provides several kinds of perspective to describe dynamic aspect of the system through interaction diagram, state diagram, and activity diagram. As a consequence, some people such as Meyer (Meyer 1997b) might find UML rather complex and inconsistent compared to BON.

In terms of completeness, UML provides more complete modeling language compared to BON. Its diagrams range from use case diagram to describe the outside view of system, until deployment diagram that is used to describe the deployment of software components on hardware nodes. Provided this kind of completeness, UML users can just simply pick up whatever they need to support their projects. The historical perspective of UML can explain why UML provides a very complete modeling language. It is obvious that during the effort to standardize object-oriented software development methods, the UML development team felt the need to accommodate various ideas coming from different methods prior to the UML.

While on one side the completeness of UML can benefit its users, on the other side it also brings some negative consequences, such as:

- The modeling language becomes too complex to understand and to use such that—as Meyer warns (Meyer 1997, 54)—instead of helping developers to think, it hinders the developer from getting to the real job.
- Not all diagrams are usable in any kinds of projects.
- Users might be trapped in the so-called "analysis-paralysis phenomenon", where they spend too much time in analysis phase by drawing too many diagrams.

Despite its completeness, UML still lacks class dictionary, which is very useful to provide a catalogue containing a sorted list of classes in the system along with its brief description. It also lacks adequate attentions to handle rules and constraints properly in order to ensure class consistency in subclassing as well as to manage reusability properly.

Advantages

Table 5-7 shows that both UML and BON have advantages in terms of object-oriented approaches. These advantages can be classified into two types:

- the advantages that are resulted from their general characteristics
- the advantages that are resulted from their modeling language and techniques.

Aside from several common advantages that can be found in both of UML and BON, each of them also provide several particular advantages that can not be found in the other.

Common advantages

The first common benefit is ***scalability***, which means the ability of a method to scale up from small textbook examples to sizeable real applications without loosing usability. This feature is very useful especially to exhibit large and complex system.

UML mainly employs package diagrams to achieve scalability. Deployment diagram can also be used for this purpose.

BON, on the other hand, achieves scalability through element compression and expansion, as well as nested clustering. As explained in Appendix D, BON provides three basic level of static description, namely system level, cluster level, and class level. By selecting a mixture of compressed and expanded forms, many different views can be obtained at various levels of details.

Another common advantage is ***seamlessness*** that guarantees a smooth transition from problem domain specification, over system design, into executable code. Object-

oriented software development regards seamless approach as the only way to result in extensive future reuse. According to Waldén and Nerson (1995, 7), "a smooth transition from user requirements over analysis and design into running system has been the goal of software engineering for over 20 years, but traditional methods (although often claiming to have the solution) have generally failed in practice".

Seamless approach becomes possible because both UML and BON use classes in all development phases. This approach, in turn facilitates requirement traceability and addresses visibility issues as well. As emphasized by Waldén and Nerson (1995, 7), "since the classes introduced in the analysis phase will still be present in the final system, tracing the propagation of initial requirements through design and implementation becomes much easier".

Both UML and BON combine seamless approach with ***reversibility*** to achieve effectiveness with respect to reuse and ease of maintenance. Reversibility means that the core elements of a notation for analysis and design should represent concepts that are directly mapped to and from an executable object-oriented language. Reversibility is important to maintain consistency between specification and implementation, as well as to promote reuse of analysis and design elements.

In a practical point of view, reversibility along with seamlessness approaches pave the way to do round-trip engineering during software development such that users may go from analysis and design to implementation back and forth.

BON strongly emphasizes seamlessness and reversibility in such a way that it only provides notations which directly support these two approaches. Hence, unlike UML, BON

does not show any endeavor to adopt several notations from other fields, such as relationship diagrams, state transition diagrams, or data flow diagrams as standard parts of its approach. BON views that these techniques will only lead to a mixed paradigm, which breaks the reversibility and introduces new complexity (Waldén and Nerson 1995, 9). In most cases it will outweigh the expected benefits. However, BON still believe that some applications may benefit from such diagrams. What BON stresses is that basing a general method on them misses the point of reversibility. Therefore, even though BON does not include such diagrams as its standard parts, BON allows its users to make use of them as complementary diagrams, if necessary.

UML, on the other hand, does not attempt to avoid such diagrams, it does support round-trip engineering through its supporting tools. Further discussion about the tools can be found in section 5.3.3.4.

UML advantages

With regard to UML, some of its diagrams and techniques provide certain benefits to its users. ***Use case diagram*** is extremely useful to capture requirements, as well as to plan and control iterative projects. This diagram is very understandable even for non-technical people, as can be seen from the result of the survey among domain experts/end-users. The appearance of actors, use cases, and the relationships between them make the diagram accurately represents the overview of the real system. Even though BON does not provide

this kind of diagram, BON embodies the concept of scenario, which is indeed similar to the concept of use case in UML.

Several other UML diagrams, namely state diagram, concurrency diagram, activity diagram, package diagram, deployment diagram, and component diagram can also be very useful in particular projects even though these kind of diagrams are not always needed in every case.

When an object performs interesting behaviors across several use cases, ***state diagram*** will become an efficacious tool to exhibit the behaviors. In the case of objects with sets of independent behaviors, ***concurrent state diagram*** plays an important role.

BON, on the other hand, does not explicitly support concurrency by providing a specific notation for concurrent modeling. There are probably two reasons for this. First reason is because when BON was introduced for the first time, the issue of object-oriented concurrency is still in its infancy. Therefore, the authors of BON decided not to invent any notation to model concurrency while it was not clearly understood what is needed in concurrent modeling. Unfortunately, up to now there is no later version of BON available to justify whether the authors of BON have changed their opinion regarding concurrency modeling. The second reason is related to BON efforts to concentrate on what in essential and needed in any kinds of projects. It is closely related to the issue of generality, which will be discussed later in this chapter.

Despite the lack of support for concurrency, BON allows its users to use complementary notations, such as state diagram, for the particular project to show more complicated concurrent behavior and process structure (Waldén & Nerson 1995, 165).

Activity diagrams are useful especially to describe complicated behaviors of an object. Since it is likely that this diagram can be easily understood by non-technical people, as shown by the result of the survey, the diagram can also be used to analyze and describe use cases in order to verify developers' understandings on use cases can with domain experts/end-users to meet their requirements. However, it is important to notice that activity diagrams do not make the links among actions and objects very clear. Even though there is a way to overcome this by using the so-called swimlanes—in which one must arrange activity diagrams into vertical zones that are separated by dashed lines; each zone represents the responsibilities of a particular class--, the offered solution sometimes is difficult to draw especially on a complex diagram (Fowler 1997, 138).

In terms of relationships, UML provides three distinctive features: composition, derived elements, and instance multiplicity. ***Composition*** is actually a stronger variety of aggregations in which the part object may belong to only one whole; further the parts are usually expected to die with the whole. Any deletion of the whole is considered to cascade to the parts. With regard to its principle of simplicity, BON on the contrary does not differentiate aggregation based on its possible variation of implementation. Instead, BON allows some freedom for modeler in using the aggregation concepts. Assertions—which are discussed later in this chapter--, plays an important role in this case.

In addition to ordinary associations and attributes, UML also provides the concept of ***derived elements*** that consist of derived associations and derived attributes. Derived associations and derived attributes are those that can be calculated from other associations and attributes, respectively. For example, a due date attribute can be derived if the borrowing

date and duration of a borrowing transaction are known. According to Fowler (1997, 82), this feature is valuable to indicate a constraint between values, and to annotate fields that are used as caches for performance reasons. BON, on the other hand, does not have this feature since BON consistently does not attempt to go into implementation details and let the programmers implement their creativity instead.

One feature that makes a significant difference between static structure of UML and static structure of BON is the existence of ***instance multiplicity*** in UML and the absence of this feature in BON. Instance multiplicity is an indication on how many objects may participate in the given relationship. It resembles the instance multiplicity from traditional data modeling. As shown in the class diagram of the case study (Appendix J), each association link has multiplicity that is indicated by numbers or asterisks. The static architecture of BON, on the other hand, does not have it. According to the authors of BON, the reasons for emphasizing instance multiplicity which simply mirrors the instance multiplicity from data modeling are historical: the multiplicity affects the way database records are designed in traditional environments (Waldén & Nerson 1995, 79). In BON's perspective, a constraint on the number of instances of a certain class is irrelevant information at the system level, and should instead be placed in the class interface specifications as part of the software contract.

BON advantages

In the general level of concepts BON strives for ***simplicity***, in a sense that it minimizes the number of concepts in the method. Following are some examples regarding this issue:

- BON provides only one kind of aggregation relationship, even though there can be several varieties of its implementation.
- There are only two basic relations between classes, namely inheritance and client relation. BON also adopts these relations to express relation between a class and a cluster, as well as between two clusters.
- BON provides only one type of grouped classes, that is clusters, even though the grouping might base on various criteria.

As a consequence of this simplicity, the author of this paper believes that BON somehow plays in higher level of approaches compared to UML. As will be discussed later, the results of the survey also indicate that it is likely that BON diagrams are preferable for system analysts/designers because they exhibit more general view of the system.

In contrast to the completeness of the UML, BON emphasizes ***generality*** as one of its characteristics. Generality means that BON concentrates on what is essential. In BON's perspective, there are three most essential things that should exist in any kind of object-oriented analysis and design projects:

1. Static architecture to describe the static structure of the system in general view
2. Formal class descriptions to clearly specify class interfaces.
3. Object scenarios to describe object behaviors

In addition to these diagrams, BON charts may also be used to enhance communication with non-technical people. However, the author of this thesis believes that the usage of these charts is optional because textual descriptions are not always preferable for users. Some users might find that graphical forms are more understandable than textual ones. The results of the survey--as discussed later in this chapter—also shows this fact.

BON, obviously does not address some diagrams that are found in UML such as state transition diagram, workflow diagram, and concurrent diagram since they are not essential enough to be used in every kind of projects. In addition, before the eyes of BON's authors they do not fit to seamless and reversibility approaches, which are strongly emphasized in BON. In case BON users need additional notation for particular projects, BON encourages them to employ one of the existing notations (Waldén and Nerson 1995, 17, 165).

The most important benefit of BON is its support to the theory of ***software contracting*** that is based on the idea of *Design by Contract™*, which was introduced by Bertrand Meyer. The idea of this theory is to use assertions to define the semantics of each class (Waldén and Nerson 1995, 10). The prerequisites and resulting behaviors of each operation are specified through pre- and post-conditions, and the overall class consistency through the class invariant. These semantic specifications then form the basis for a contract between each class; the supplier, and its client classes. In this theory, a software system is best viewed as a network of cooperating clients and suppliers whose exchange of requests and services are precisely defined through decentralized contracts.

BON support to software contract is achieved through a ***fully-typed notation*** for class interfaces. As can be seen in formal class descriptions (see BON model of the case study in Appendix K), a class interface consists of syntax and semantics of its operations. In these two features (i.e. software contracting and typed-interface description), indeed, lay the core of BON method that makes it different from UML.

Software contracting is very important to build clear interfaces, to maintain consistency in subclassing, as well as to guarantee proper reusability of the software components. A very famous example on the impact of neglecting software contracting in software development project implementing reusable components is the case of Ariane 5 which caused loss of half a billion dollars[1].

Despite the value of software contracting, UML does not talk much about it. However, according to Fowler (1997, 73), one can use them in UML without any trouble. Furthermore, Fowler explains that invariants are actually equivalent to constraint rules on class diagrams, whereas operation pre-conditions and post-conditions can be documented within the operation definition.

Through its support to software contracts, BON actually paves the way to the creation of more intelligent CASE tools since in the fully-typed of interface descriptions each class is specified clear enough to be automatically converted into a real code. Unfortunately, even though BON has claimed its contribution as a basis for intelligent

---

[1]Further description on this case is available at htp://eiffel.com/doc/manuals/technology/contract/ariane/page.html

CASE tools, the only CASE tool supporting BON does not even provide a proper support to assertions[2].

Aside from providing very detail description of classes and its operations through its fully-typed class interfaces, BON also provides a very ***high-level untyped view*** in a form of informal charts. CRC cards that were introduced by Ward Cunningham and Kent Beck inspire the use of these charts. Theoretically, in the very early of the process these charts can be used to communicate basic ideas to non-technical people, such as domain experts and end-users. Developers can make us of these charts, especially class chart, to capture users' requirements by letting users to fill out the charts. As suggested by Fowler (1997, 66), class chart may also encourage animated discussion among the developers. Furthermore, Fowler says that the use of class chart will be good for the team to avoid getting bogged down into too many details too early as well as to avoid cluttered and unclear definitions of classes. In the later phases, the charts may also serve as very high-level of documentation for the system. Surprisingly, the survey shows that four out of five domain experts/end-users found cluster chart is neither understandable nor usable. Furthermore, they said that textual form like BON charts are not more understandable than graphical forms. Some of them commented that the terms used in cluster chart are confusing. Nevertheless, system analysts/designers and programmers tend to perceive that BON charts are understandable, usable, and effective.

---

[2]http://eiffel.com/services/userlist/eiffelcase_and_bon.html

In the detail level of concepts, BON has one distinctive relation between **classes,** namely shared association and a specific feature in terms of static relations, that is role multiplicity. These two features--shared association and role multiplicity--, can be very useful to express some special situations that might happen in the real world.

Shared association is a special case of association in which whenever an instance of the client class is attached to an instance of the supplier class, it will always be to the same supplier instance (or one in a fixed set). It is useful to represent a situation where a group of related objects shares a common supplier, which provides a certain service to the group. A common example of shared association is a set of personal computers sharing a common server.

In some other cases, there can be several relations between the same two classes A and B. To represent these situations, BON provides the so-called ***role multiplicity***, which can be expressed either using a multiplicity marker or attaching several labels to the same graphical link. An example of this situation can be found in the case study, in which there are three relations between `ITEM_COPY` and `BORROWER`, namely *holder, reserver,* and *borrower.* In the static architecture of BON, this fact is easily represented by attaching three labels to the graphical link between `ITEM_COPY` and `BORROWER`. By doing so, in the later stage it will help designer to implement these relations in proper features of classes. In contrast to this, the class diagram of UML does not contain any information that there are actually three relations between `ITEM_COPY` and `BORROWER`. Consequently, the designer should somehow keep this fact in mind, especially during the detail design. A problem might arise

when the designer team overlooks this fact because there is no documentation containing the information about the three relations.

5.3.3.2 Understandability, Usability, and Effectiveness of the Modeling Language

Based on the result of the survey, table 5-8 in the following is created with regard to the detail comparison criteria.

Table 5-8 Comparison on Understandability, Usability, and Effectiveness of the Modeling Language

NO	CRITERIA	BON	UML
1.	Understandable?	+ It seems that all BON charts and diagrams are understandable for system analysts/designers and programmers. – Static architecture for domain experts/end-users: • 2/5 : understandable • 3/5 : not understandable – It is likely that cluster chart is not understandable for domain experts/end-users – For most respondents textual descriptions are not more understandable than graphical ones	+ It seems that almost all of UML diagrams are understandable for system analysts/designers & programmers, except deployment diagram for system analysts/designers: • 5/10 : understandable • 5/10: not understandable + It seems that the following diagrams are understandable for domain experts/end-users: • Use case diagram • Class diagram • Activity diagram – Activity diagram: links among actions & objects are not clear
2.	Usable?	+ It seems that all BON diagrams and charts are usable for system analysts/designers and programmers	+ It seems that the following diagrams are usable for system analysts/designers and programmers: • Use case diagram

		+ It is likely that static architecture is usable for domain experts/end-users: – It is likely that cluster chart is not usable for domain experts/end-users	• Class diagram • Collaboration diagram • Activity diagram • State diagram • Deployment diagram + It seems that the following diagrams are usable for domain experts/end-users: • Use case diagram • Class diagram • Activity diagram – Package diagram for system analysts/ designers: • 5/10 : usable • 5/10 : not usable – Package diagram for programmers: • 1/7 : usable • 4/7 : not usable
3.	Effective?	+ It seems that all BON diagrams and charts are effective for system analysts/designers and programmers	+ It seems that the following diagrams are effective for system analysts/designers and programmers: • Use case diagram • Class diagram • Collaboration diagram • Activity diagram • State diagram • Deployment diagram – About half of system analysts/designers and programmers perceive that package diagram is effective, while the other half think that it is not.
4.	Preferable?	+ Half of system analysts/ designers prefer static	+ All domain experts/end-users prefer class diagram

		architecture and formal class descriptions – Only 2/7 programmers prefer static architecture and formal class descriptions + About half system analysts/ designers prefer object scenario + Half of programmers perceive that object scenario is more understandable – None of programmers perceive that object scenario is more effective for them	+ About half of system analysts/designers prefer class diagram + Most programmers prefer class diagram – Only a few of system analysts/ designers prefer collaboration diagram + Half of programmers perceive that collaboration diagram is more understandable + Most programmers think that collaboration diagram is more effective for them

With regard to the survey, it can be seen that BON cluster chart is neither understandable nor usable for 4 out of 5 domain experts/end-users. Further, according to 4 out of 5 domain experts/end-users and 7 out of 10 system analysts/designers textual descriptions like BON charts are surprisingly not more understandable compared to graphical descriptions. However, 4 out of 7 programmers perceive that textual descriptions are more understandable. This finding is quite different from what BON has been claimed that its textual charts are useful to enhance communication with non-technical people.

The survey also shows a general tendency that most system analysts/designers and programmers perceive that all BON charts and diagrams are understandable, usable, and effective. Most of them also think that almost all UML diagrams, except package diagram and deployment diagram, are also understandable, usable, and effective.

In the case of ***package diagram***, more or less half of system analysts/designers as well as programmers believe that it is usable and effective while more or less the half remaining believe that it is not usable and not effective either. The author of this thesis however believes that package diagram will be very useful in a large system in which it is difficult to exhibit the whole system in a piece of paper. Considering that the case study that is chosen for this study is relatively small and less complicated, users might overlook the important of package diagrams. Therefore, they might think that the diagram is not usable.

In the case of ***deployment diagram***, around half of system analysts/designers perceive that it is understandable and effective, whereas the other half do not think it is. However, 6 out of 10 of them believe that deployment diagram is usable. Most programmers, on the other hand, perceive that this diagram is understandable, usable, and effective for them.

Even though the general trend shows that respondents assess all diagrams of BON and UML as understandable for them, they have preferences regarding the similar diagrams that are provided by both BON and UML.

With regard to ***static model***, 5 out of 10 system analysts/designers prefer static architecture with its formal class description than class diagram because the diagrams provide them with a general level description of system that is suitable to their job as analysts/designers. However, 4 out 10 prefer class diagram instead.

On the other hand, 5 out of 7 programmers prefer class diagram rather than static architecture because they believe that class diagram is more efficient to help them during implementation stage. This finding emphasizes the above-mentioned hypothesis that BON

diagrams provide higher level descriptions compared to UML diagrams. However, it is important to notice that the class diagram, which is included in the questionnaire for system analysts/designers and programmers, is the class diagram from specification perspective. The one from conceptual perspective (see Appendix J)—which is more general or high level—is not shown to these groups of respondents. Therefore, it is not known whether system analysts/designers will still prefer static architecture over class diagram if they were asked to compare static architecture with class diagram from conceptual perspective. As a matter of fact, non-technical people— who are asked to compare these last two diagrams—prefer class diagram rather than static architecture.

Despite programmers' preferences towards class diagram, the author of this thesis suspects that respondents' unfamiliarity with Eiffel language—to which BON is closely associated--, affect their perceptions towards the efficiency of the diagram during implementation. In fact, all respondents are familiar with Java and some of them are also familiar with C++ that are directly supported by UML. Therefore, it is possible that if the respondents were familiar with Eiffel, they might have different opinion regarding the diagrams. However, this hypothesis needs further research.

In regard to ***dynamic model***, more or less half of ten system analysts/designers observe that object scenario is more understandable and more effective than collaboration diagram.

Despite the fact that the number of programmers who think that object scenario is more understandable compared to collaboration diagram is the same with the number of programmers who think the other way around, most programmers obviously prefer

collaboration diagram rather than object scenario because they perceive collaboration diagram is more effective to help them during implementation. Again in this finding, it can be seen that BON diagrams are more general than UML diagrams.

The results of this survey, however, shows only general trends among a limited number of respondents. Therefore, the results still need further verification through a more formal survey.

5.3.3.3 Process

Table 5-9 in the following presents the results of phase 4 with regard to the process.

Table 5-9 Comparison on process

NO	CRITERIA	BON	UML
1.	Well-defined process?	+ Well-defined process	+ Well-defined process
2.	Iterative process?	+ Iterative process	+ Iterative process
3.	Configurable process?	+ Configurable process	+ Configurable process
4.	Advantages?		+ Full support to SDLC + Managerial aspects

As explained in Chapter IV, UML has different point of view from BON with regard to software development process. BON explicitly includes process as one of its parts, aside from its modeling language. UML, on the other hand, does not include process as its core parts since--as reflected by its name--, its focus is on modeling language. Software development process supporting UML called Rational Unified Process is actually an extension of the UML. However, since the purpose of Rational Unified Process is to allow

development teams to gain full benefits of UML[3], for the purpose of this study BON process is compared to Rational Unified Process.

Both Rational Unified Process and BON process provide a ***well-defined advised process*** as a guideline for development team through critical software development activities. These processes are ***iterative*** and ***configurable***.

Presented in a two-dimension diagram (Appendix C), Rational Unified Process describes a clear and understandable process in terms of dynamic as well as static aspects. From the diagram, it is also obvious that Rational Unified Process is iterative, fulfilling the need of it in object-oriented software development.

BON defines 9 major tasks and 9 major activities to complete, along with clear descriptions of what are needed and what can be expected from each task, as well as how to measure the acceptance of each fulfilled task.

BON process does not either overlook the need of iterative process by emphasizing that although its tasks are often listed in sequential order, they are usually iterative in practice (Nerson 1992, 66).

Realizing that no single process is suitable for all software development, both Rational Unified Process[4] and BON (Waldén and Nerson 1995, 148) process provide such flexibility for users to configure the process in order to help them pursuing their goals.

---

[3]http://www.rational.com/uml/whitepapers

[4]http://www.rational.com/uml/whitepapers

Users, of course, will get a benefit from a configurable process since they can easily modify the process based on the nature of their projects.

Rational Unified Process provides a full ***support to SDLC***. As can be seen in figure C-1 (Appendix C), its organization along content consists of core process workflows and core supporting workflows. Core process workflows explicitly include planning phases until implementation phase of SDLC. Figure C-2 of Appendix C shows that Rational Unified Process also supports the last phase of SDLC, that is support phase. This is done by implementing an iterative process consisting of the same workflows after performing an evaluation to what have been accomplished. BON process, on the other hand, only supports particular parts of systems development life cycle (SDLC), namely analysis and design.

Rational Unified Process also pays adequate attentions to ***managerial aspects*** of software development projects. It includes three core supporting workflows that are closely related to management, namely configuration and change management, project management, and environment, in addition to business modeling workflow.

Thus, compared to BON process that merely focuses on analysis and design phases of SDLC with regard to technical aspects of a software project, Rational Unified Process offers a more comprehensive perspective towards software development projects, which in turn give benefits to its users.

5.3.3.4 Environment

Table 5-10 in the following presents the results of phase 4 with regard to environment.

Table 5-10 Comparison on environment

NO	CRITERIA	BON	UML
1.	Existing CASE tools?	+ One existing CASE tool + An integrated software development environment	+ Two existing CASE tools + An integrated software development environment
2.	Successful projects?	+ Number of successful projects	+ Number of successful projects
3.	Supporting users?	+ Number of supporting users + Former users of UML who were dismayed because of UML's complexity	+ Number of supporting users
4.	Literatures?	– Lack of available literatures	+ Abundant of literatures

There are at least two existing CASE tools that support UML, namely Rational Rose® and Object Team®. The first one is a product of **Rational Software**—the main sponsor of UML--, whereas the other came from **Interactive Software Engineering Inc. (ISE)**—the main sponsor of BON—in cooperation with **Cayenne**.

Aside from Rational Rose® whose main purpose is for extensive use in visual modeling, **Rational Software** also offers a full suite of Rational Development tools that addresses the full software development life cycle, including:

- Requisite®Pro : for requirements management

- Rational Rose® : for visual modeling
- ClearCase® : for configuration and change management
- ClearQuest™ : for change request management
- SoDA® : for automated documentation
- TeamTest® : for automated testing
- RationalApex® : for Ada and C++ development.

All of these tools are integrated in the Rational Unified Process to enable efficient development of high-quality enterprise applications by providing mechanism to do round-trip engineering. The tools particularly support C++, Java, and Ada. Object Team®, on the other hand is built particularly for the users of UML who want to implement their designs using Eiffel language.

On the other hand, EiffelCase® is the only CASE tool to support BON. However, ISE offers a software development environment called ISE Eiffel®. According to the announcement that is posted in Eiffel homepage[5], this environment provides an integrated object-oriented solution for software developers, from analysis and design to code generation, maintenance, and reverse engineering. Figure 5-7 describes the components of ISE Eiffel®. This environment particularly supports **Eiffel** language that directly implements the idea of software contracting in its semantics.

[5] http://eiffel.com/products/page.html

With regard to ISE Eiffel® generally and EiffelCase® particularly, some users found that the tools still need some enhancements to really achieve the level of round-trip engineering that has been announced. Jim Nelson in ISE Eiffel discussion forum[6] once brought up a topic about EiffelCase® and BON. As one of EiffelCase® users who believes that EiffelCase® is very attractive for Eiffel development, he gives some suggestions for future enhancement of EiffelCase®. According to Nelson, EiffelCase® has some missing features such as lack of support to BON charts, assertion language, dynamic model, as well as automatic generated code. In addition, Nelson encourages an integration of EiffelCase® with EiffelBench®—a workbench supporting Eiffel--, in order to provide Eiffel users a greater level of seamlessness during development.

However, it is beyond this study to practically evaluate the tools. Another study may then be performed for this purpose.

From the information available in the Internet, it can be seen that both UML and BON have a quite number of followers. Each of them also has been proven in a quite number of successful projects. In regard to UML, there is no available information on the number of successful projects. However, UML has been formally accepted as an industry standard with endorsement from at least 18 worldwide companies'[7].

---

[6]http://eiffel.com/services/userlist/eiffelcase_and_bon.html

[7]http://stud2.tuwien.ac.at/~8726711/ummw2#MU3

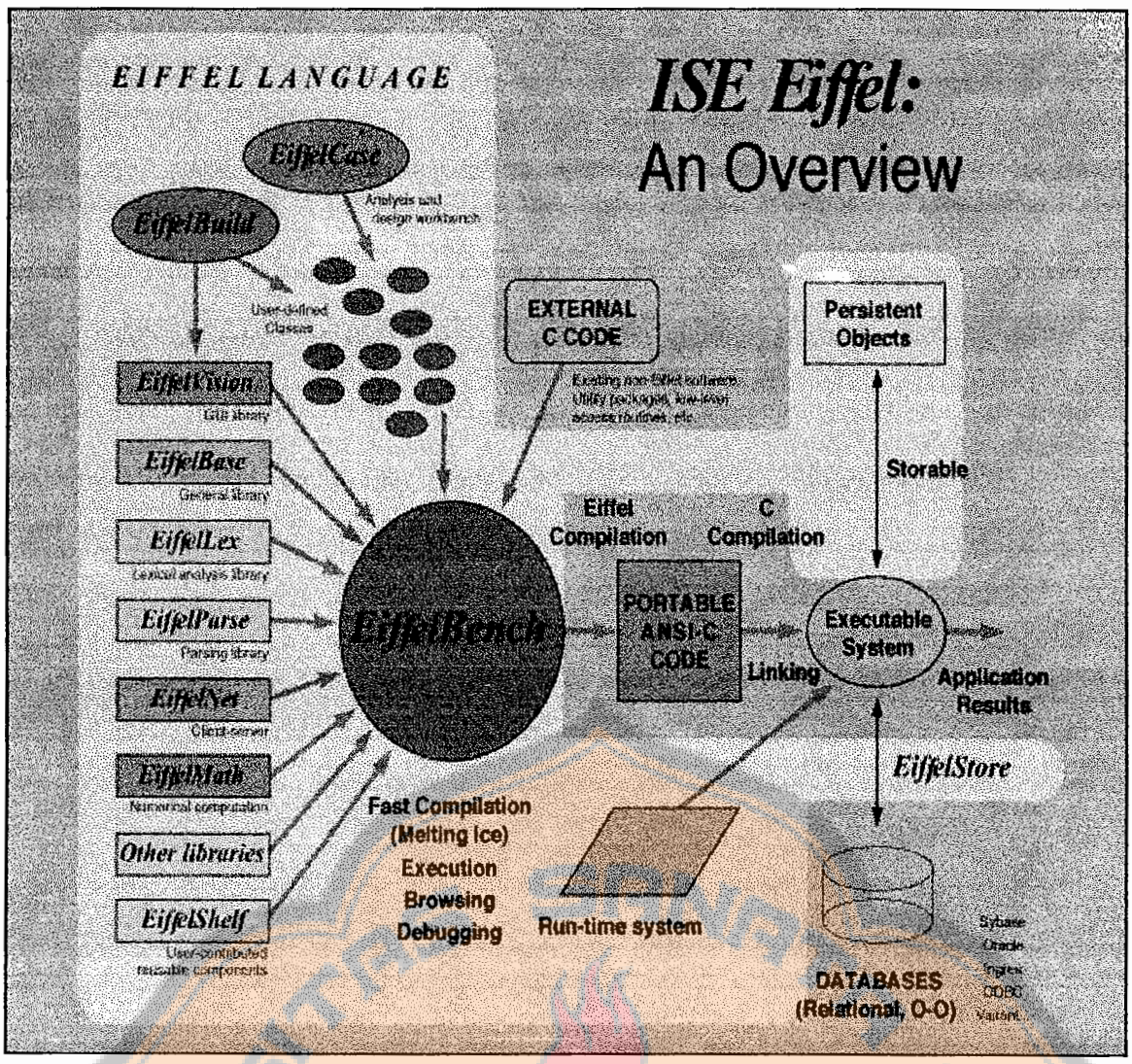


Figure 5-7 ISE Eiffel Environment
(Source: http://eiffel.com/products/page.html)

There are at least 43 projects that have been done using BON[8]. A cyberspace group discussion called "ISE Eiffel discussion forum" with some numbers of active participants

[8]http://eiffel.com/eiffel/projects/list.html

can also be found in http://eiffel.com/eiffel/services/userlist. The topic discussed in this group is not merely about BON but also about Eiffel, the programming language closely related to BON--. From this discussion group, it can be found that one user (perhaps more) of Eiffel and BON is actually a former user of UML. He switched from UML to BON because of the complexity of UML[9].

With regard to UML, there are also a lot of literatures available in the market. In contrast, only few literatures on BON are available. As a matter of fact, there is only one book by Waldén and Nerson (1995) that provides an extensive explanation on BON.

[9]http://eiffel.com/services/userlist/eiffelcase_and_bon.html

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

1. Table 6-1 on the following page presents the summary of advantages and disadvantages of UML and BON in terms of general criteria
2. The main difference between UML and BON lays on BON support to the concept of software contracting. This important and useful feature is unfortunately not formally supported by UML.
3. Despite all the differences, the core/essential modeling language of UML and BON have some similarities.
4. Both UML and BON are open to the utilizing of other existing notation if necessary.

Table 6-1 Advantages and disadvantages of UML and BON

CRITERIA	UML		BON	
	ADVANTAGES	DISADVANTAGES	ADVANTAGES	DISADVANTAGES
OO APPROACHES	1. Complete underlying concepts 2. Complete diagrams 3. Scalable 4. Seamless 5. Reversible 6. Use case diagram	1. Complex diagramming 2. Mix essential and optional diagrams 3. No support to software contracting	1. Complete underlying concepts 2. General & simple 3. Scalable 4. Seamless 5. Reversible 6. Support to software contracting through formal class description 7. Class dictionary	Needs complementary notations for particular projects
UNDERSTANDABILITY, USABILITY, & EFFECTIVENESS OF MODELING LANGUAGE (with regard to particular groups of respondents)	1. Most diagrams are understandable, usable, and effective 2. Following diagrams are preferable • Class diagram: for non-technical, programmers, & half of system analysts/designers • Collaboration diagram: for programmers	Not all diagrams are usable for every kind of projects	1. All *diagrams* are understandable, usable, & effective 2. High-level diagramming approach: preferable for half of system analysts/ designers	1. BON textual charts are not more understandable than graphical descriptions 2. Less preferable for domain experts/ end-users and programmers

PROCESS	1. Well-defined 2. Iterative 3. Configurable 4. Full support to SDLC 5. Address managerial aspects		1. Well-defined 2. Iterative 3. Configurable	1. Partial support to SDLC 2. No support to managerial aspects
ENVIRONMENT	1. Industry standard 2. Software development environment 3. Number of users 4. A lot of literatures	Some dismayed users	1. Software development environment 2. Number of users	Few literatures

6.2 Recommendations

1. By recognizing every strengths and weaknesses of UML and BON, users may decide which one will be adopted. In this context, there are three alternative solutions that might be taken. Table 6-2 describes each alternative along with its corresponding cautions and suggestions. The table also elaborates for what/whom it is suitable. The term "ones" that is used in the table refers not only to organizations that are adopting object-oriented technology but also educational institutions that are offering courses in object-oriented technology. Whereas the term "projects" refers to individual/school projects and organizational projects as well.

2. The author of this thesis strongly recommends the third alternative solution because by implementing this combination users can take advantage from the strengths of each method and minimize the effects of their weaknesses. UML as well as BON make the combination possible since both of them are open to the utilizing of other existing notations.

3. Based on the third alternative solution, a modified model of the Book Circulation System can be built. The model consists of the following chart and diagrams: (1) Use case diagram; (2) Activity diagram; (3) Class chart; (4) Class diagram from conceptual

perspective; (5) Class diagram from specification perspective; (6) Formal class description; (7) Collaboration diagram; and (8) Class dictionary.

4. For a further study regarding BON and UML, the author of this thesis suggests:
   - A more formal and comprehensive survey that addresses not only the modeling language but also the underlying concepts, the process, and the techniques.
   - To select respondents who know Java/C++ and Eiffel for the survey.
   - To perform a more practical comparative study on the supporting tools.

Table 6-2 Alternative Solutions

NO	METHOD	SUITABLE FOR	CAUTION	SUGGESTION
1.	BON	• Ones who like simplicity • Introduction course in OOAD • Projects with very much concerns on reusability • Projects that will be implemented using Eiffel	BON lacks several auxiliary notations & supporting literatures	Employ complementary notations if needed
2.	UML (ALONG WITH RATIONAL UNIFIED PROCESS)	• Ones who need guarantees from the widespread used and the availability of literatures, as well as ones who want to maximize the advantage of Rational Unified Process in terms of project management. • Introduction course in OOAD: essential diagrams • Advance course in OOAD: optional diagrams • Projects that particularly will be implemented using Java, C++, or Ada	Be careful in discriminating and choosing which notations are applicable for a certain problem	1. Separate UML diagrams into: • *Essential diagrams:* use case diagram, class diagram, interaction diagram • *Optional diagrams:* activity diagram, state diagram, package diagram, component diagram, deployment diagram 2. Fowler (1997) provides a good guidance for UML users to decide which diagram is needed to address particular problems in modeling.

3.	COMBINATION OF BON & UML	• Ones who like to get benefits from UML and BON. • Advance course in OOAD: – Needs creativity & critical point of view – May lead to a "new/revised method" and an improved CASE tool • Projects that need the strengths of UML and BON, with an adequate attention on project management.	Be careful with the underlying concepts of the combination of these two methods.	1. Figure 6-1 provides an illustration to this combination. 2. Table 6-3 elaborates the reasons for including each element in the combination 3. Rational Unified Process (with some enhancements) may serve as its corresponding process 4. Object Team® might be considered as the related CASE tool because it supports UML-Eiffel

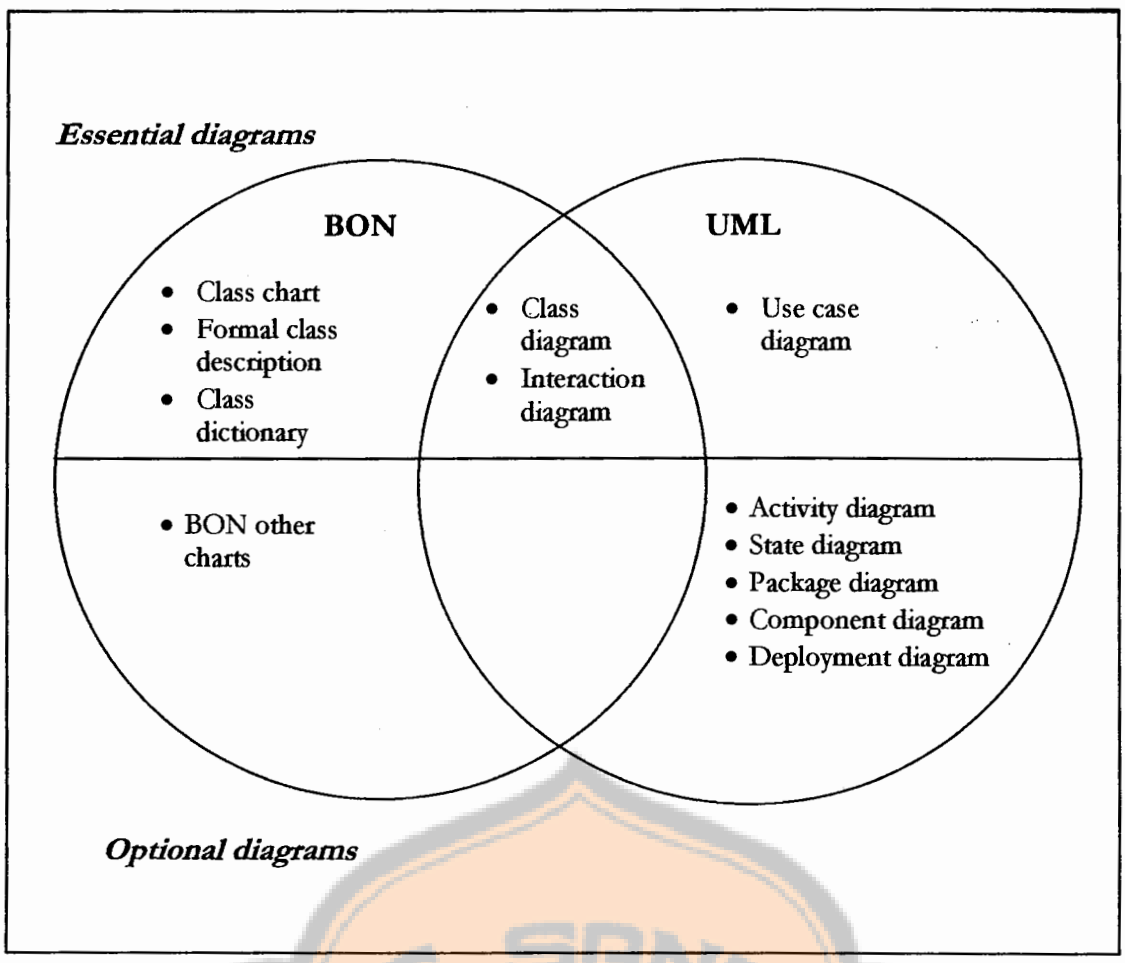


Figure 6-1 Combination Modeling Language

Table 6-3 Elements of Combination Modeling Language

a. Essential diagrams

NO	DIAGRAM	SOURCE	REASONS
1.	Class chart	BON	• Essential to identify elements of a class in terms of queries, commands, and constraints • Useful as a tool to capture users' requirements • Encourage animated discussion among the developers
2.	Formal class description	BON	• Effective to build clear class interfaces • Effective to identify rules and constraints of a class • Effective to maintain consistency in subclassing • Guarantee proper reusability • It seems understandable, usable, effective according to the respondents
3.	Class dictionary	BON	Essential to provide a catalogue containing a sorted list of classes in the system along with their brief descriptions
4.	Use case diagram	UML	• Effective to capture users' requirements, • Effective to plan and to control iterative projects. • It seems understandable for technical and non-technical people
5.	Class diagram	UML	• Essential to describe the static structure of the system. • It seems preferable compared to static architecture.

6.	Interaction diagram	UML	• Essential to describe the dynamic behaviors of the system. • It seems understandable & effective for system analysts/designers & programmers • It seems more effective to help during implementation than object scenario.

b. Optional diagram

NO	CHART/DIAGRAM	SOURCE	REASONS	USED IN THESE PARTICULAR CONDITIONS
1.	BON other charts (system chart, cluster chart, event chart, scenario chart, object creation chart)	BON	• Can be used to enhance communication with non-technical people • Can be used as complementary descriptions to the graphical model	When non-technical people lacks of ability to understand graphical descriptions
2.	Activity diagram	UML	• Can be used to show behavior within control structure • Can be used to analyze and describe a use case • Can be used to describe implementation of a method	• When an object perform complicated or parallel behaviors • When there is a need to analyze and describe a use case in a graphical description

3.	State diagram	UML	Can be used to exhibit behaviors of an object across several use cases	• When an object perform an interesting behavior • When an object has a set of independent behaviors (use concurrent state diagram)
4.	Package diagram	UML	Can be used to show groups of classes and dependencies among them	When the system is too complex and too large to be depicted in a piece of paper.
5.	Deployment diagram	UML	Can be used to show deployment of software components on hardware nodes	When the system involves distributed concepts

APPENDIX A: UML DIAGRAMS

NAME	DESCRIPTION
USE CASE DIAGRAM	Shows the relationship among actors and use cases within a system.
CLASS DIAGRAM	• Shows static structure of concepts, types, and classes. • Concepts show how users think about the world; types show interfaces of software components; classes show implementation of software components.
STATE DIAGRAM	Shows the behavior of a system involving all possible states a particular object can get into and how the object's state changes as a response to the receipt of outside stimuli.
ACTIVITY DIAGRAM	• Shows behavior with control structure. • Can show many objects over many uses, many objects in single use case, or implementation of method. • Encourages parallel behavior.
INTERACTION DIAGRAM	• Shows how several objects collaborate in single use case. • There are two forms: SEQUENCE DIAGRAM and COLLABORATION DIAGRAM. • Sequence diagrams show an interaction arranged in time sequence. • Collaboration diagrams show the relationships among objects.
IMPLEMENTATION DIAGRAM	• Shows aspects of implementation, including source code structure and run-time implementation structure. • It comes in two forms: COMPONENT DIAGRAM and DEPLOYMENT DIAGRAM. • Component diagrams show the dependencies among software components. • Deployment diagrams show physical layout of components on hardware nodes.

Use Case Diagrams

A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and the use cases, and generalizations among the use cases. A use case diagram shows the relationship among actors and use cases within a system.

Use case diagrams were generated from the Jacobson method that relies on use cases as a way to elicit classes. Fowler (1997, 43) defined a use case as a typical interaction between a user and a computer system. While Fowler prefers the term users, Jacobson uses the term *actors* for users of the future system (Meyer 1997a, 738). However, both of them refer to the same set of person who will be using the system.

Use case diagrams may contain several standard relationships among use cases or between actors and use cases. These are the following:

1. *Communicates the* participation of an actor in a use case. This is the only relationship between actors and use cases.
2. *Extends*: An extends relationship from use case A to use case B indicates that an instance of use case B may include the behavior specified by A.
3. *Uses*: A uses relationship from use case A to use case B indicates that an instance of the use case A will also include the behavior as specified by B.

Class Diagrams

Class diagrams of UML are a melding of OMT, Booch, and class diagrams of most other object-oriented methods. Some extensions can be defined for various diagrams to support other modeling styles.

Class diagrams show the static structure of the model, in particular, the things that exist (such as classes and types), their internal structure, and their relationships to other things. A class diagram may also contain interfaces, packages, relationships, and even instances, such as objects and links. In another way, a class diagram describes the types of objects in the system, the various kinds of static relationships that exist among them, the attributes and operations of a class and the constraints that apply to the way objects are connected (Fowler 1997, 53).

There are several kinds of static relationships that may exist among the objects in the system. These relationships can be classified into two principal static relationships, namely:

1. *Associations:* represents relationships between instances of classes.
2. *Subtypes*: refers to the relationship between an object and its type.

Subtype relationships consist of several variations, such as *classification, aggregation, composition*, and some others that can be derived from these three.

State Diagram

A state machine is a graph of states and transitions that describes the response of an object of a given class to the receipt of outside stimuli. A state machine is attached to a class or a method.

A statechart diagram represents a state machine. It shows the behavior of a system involving all the possible states a particular object can get into and how the object's state changes as a result of events that reach the object (Fowler 1997, 121). The states are represented by state symbols, whereas arrows connecting the state symbols represent the transitions of an object state.

Activity Diagrams

An activity model is a variation of a state machine in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. According to Fowler (1997, 129), the activity diagram is a combination of several techniques prior to UML, namely the event diagrams of Jim Odell, SDL state modeling techniques, and Petri nets.

Interaction Diagrams

An interaction diagram shows the pattern of interaction among objects. There are two forms of interaction diagrams, namely *sequence diagram* and *collaboration diagram*. Even though both of them based on the same underlying information, each emphasizes a particular aspect of it.

A sequence diagram shows an interaction arranged in time sequence]. In particular, it shows the object participating in the interaction by their "lifelines" and the messages that they exchange arranged in time sequence. It does not show the associations among the objects. Collaboration diagrams, on the other hand, show the relationships among objects.

The choice of which form of interaction diagram to use depends on the preferences of each developer (Fowler 1997, 11).

Implementation Diagrams

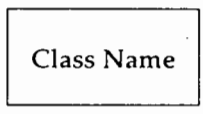
Implementation diagrams show aspects of implementation, including source code structure and run-time implementation structure. They come in two forms: *component diagrams* and *deployment diagrams*.

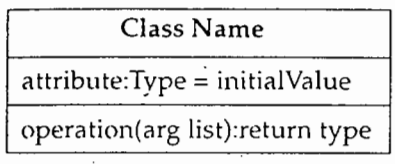
A component diagram shows the dependencies among software components, including source code components, binary code components, and executable components.

Deployment diagrams show the configuration of run-time processing elements and the software components, processes, and objects that live on them.

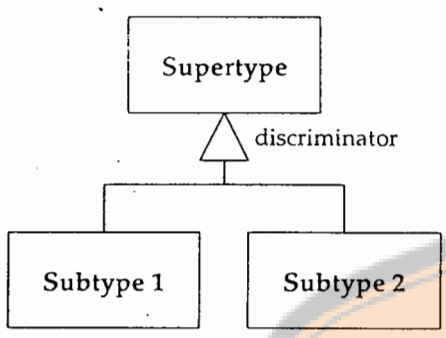
APPENDIX B: UML NOTATIONS

Class





Generalization



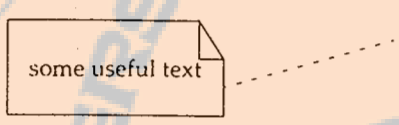
Constraint

{description of constraint}

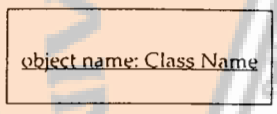
Stereotype

«stereotype name»

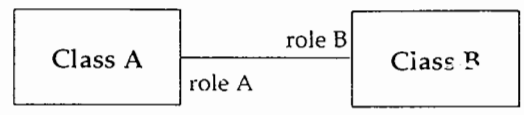
Note



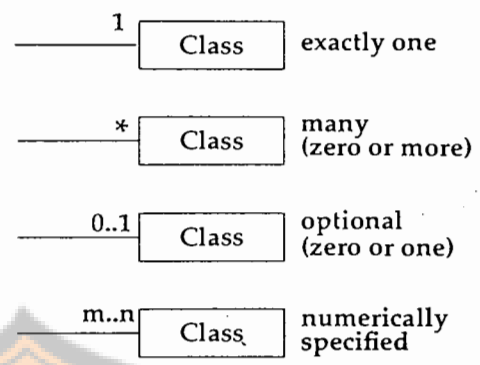
Object

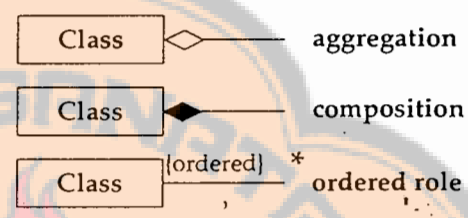


Association

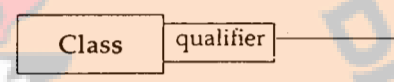


Multiplicities

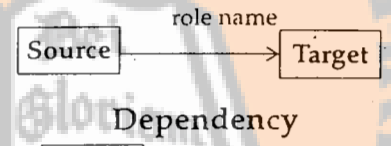




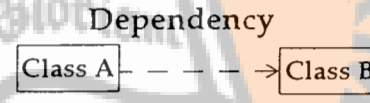
Qualified Association



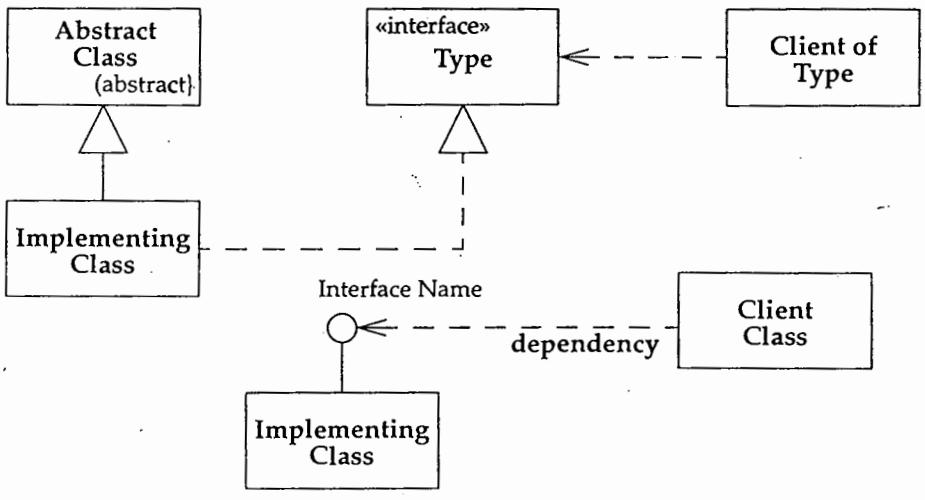
Navigability



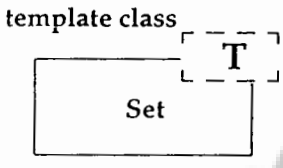
Dependency



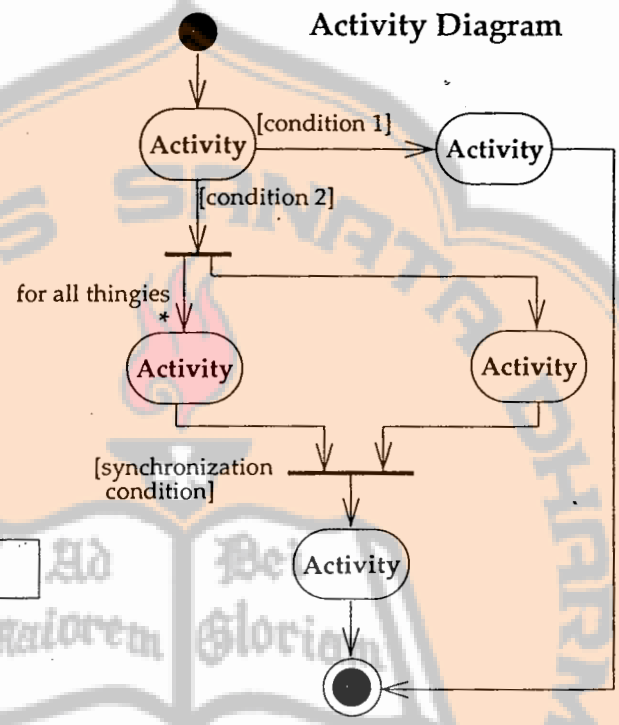
Class Diagram: Interfaces



Class Diagram: Parameterized Class

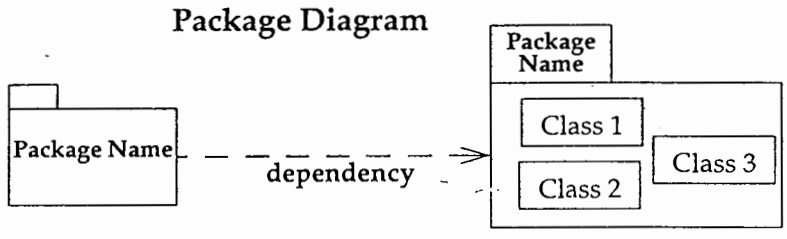


Activity Diagram

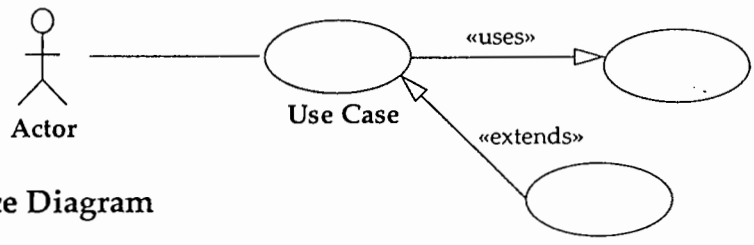


Association Class

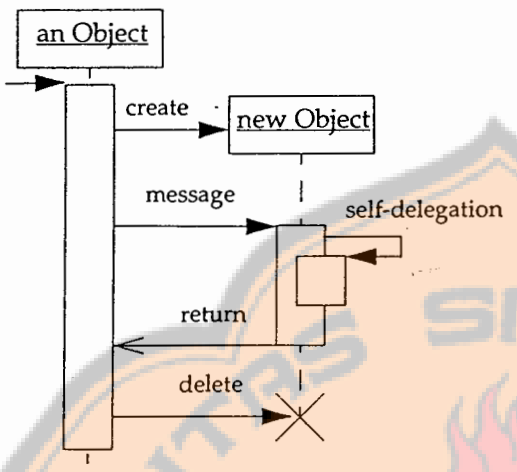
Package Diagram



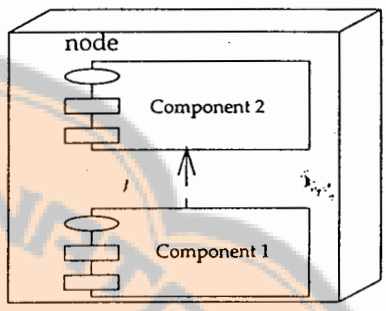
Use Case Diagram



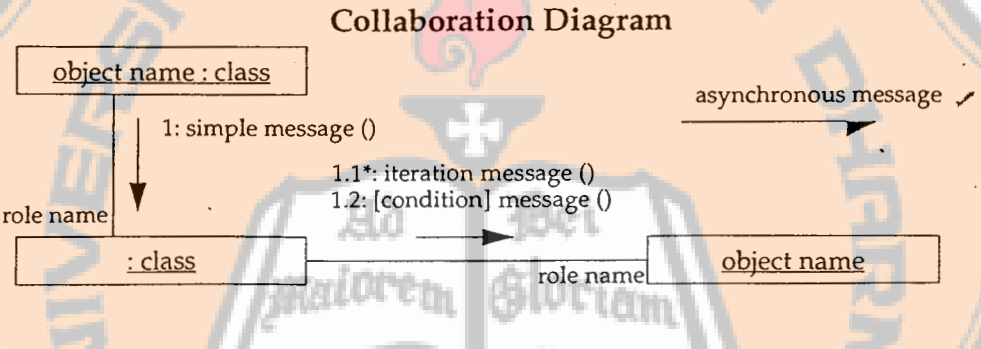
Sequence Diagram



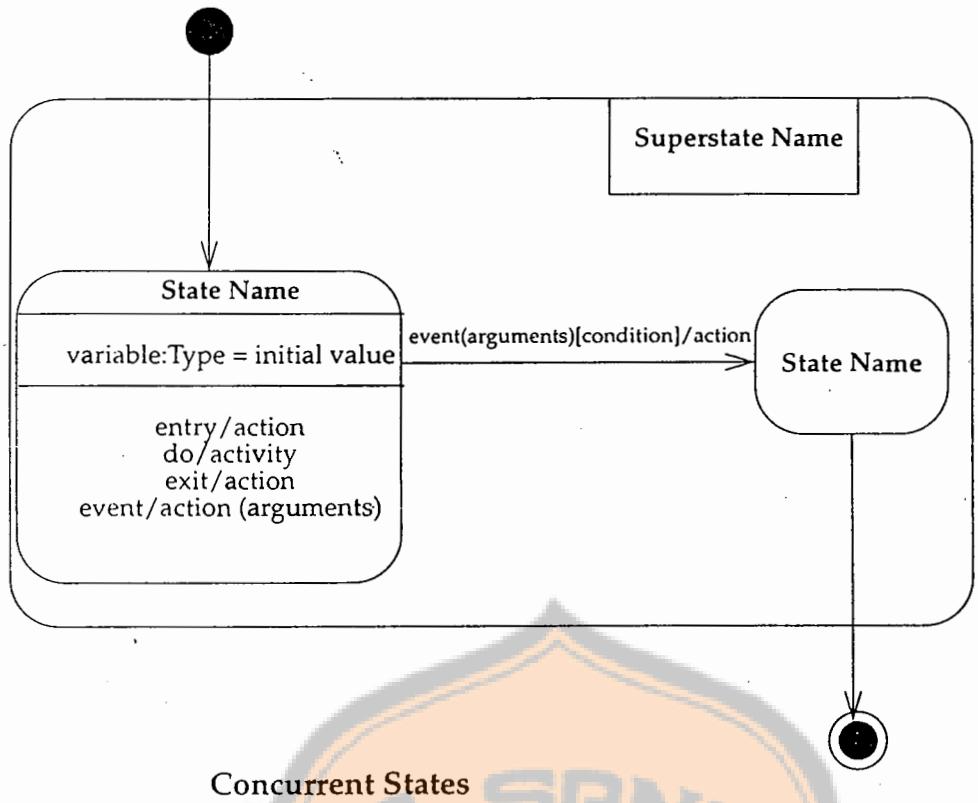
Deployment Diagram



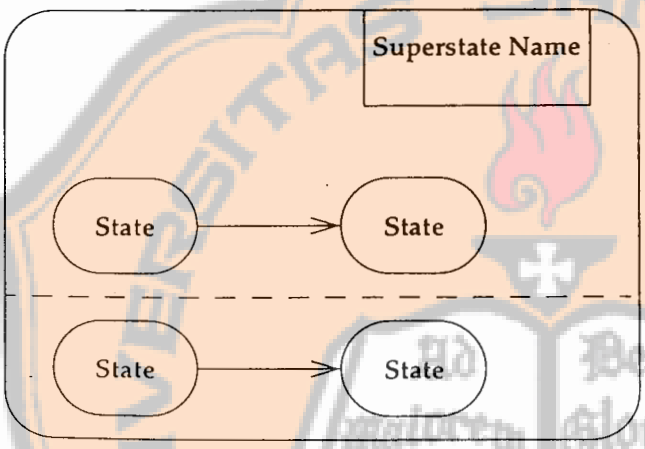
Collaboration Diagram



State Diagram



Concurrent States



APPENDIX C: RATIONAL UNIFIED PROCESS

Organization along Time

As can be seen in figure C-1, the software lifecycle is broken into *cycles*, each cycle working on a new generation of the product. One development cycle is divided into four consecutive *phases*:

1. *Inception phase*: specifying the project vision
2. *Elaboration phase:* planning the necessary activities and required resources; specifying the features and designing the architecture.
3. *Construction phase:* building the product as a series of incremental iterations.
4. *Transition phase:* supplying the product to the user community (manufacturing, delivering, and training).

Each phase is concluded with a well-defined *milestone*—a point in time at which certain critical decisions must be made, and therefore key goals must have been achieved. Each phase can be further broken down into *iterations*. An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows incrementally from iteration to iteration to become the final system.

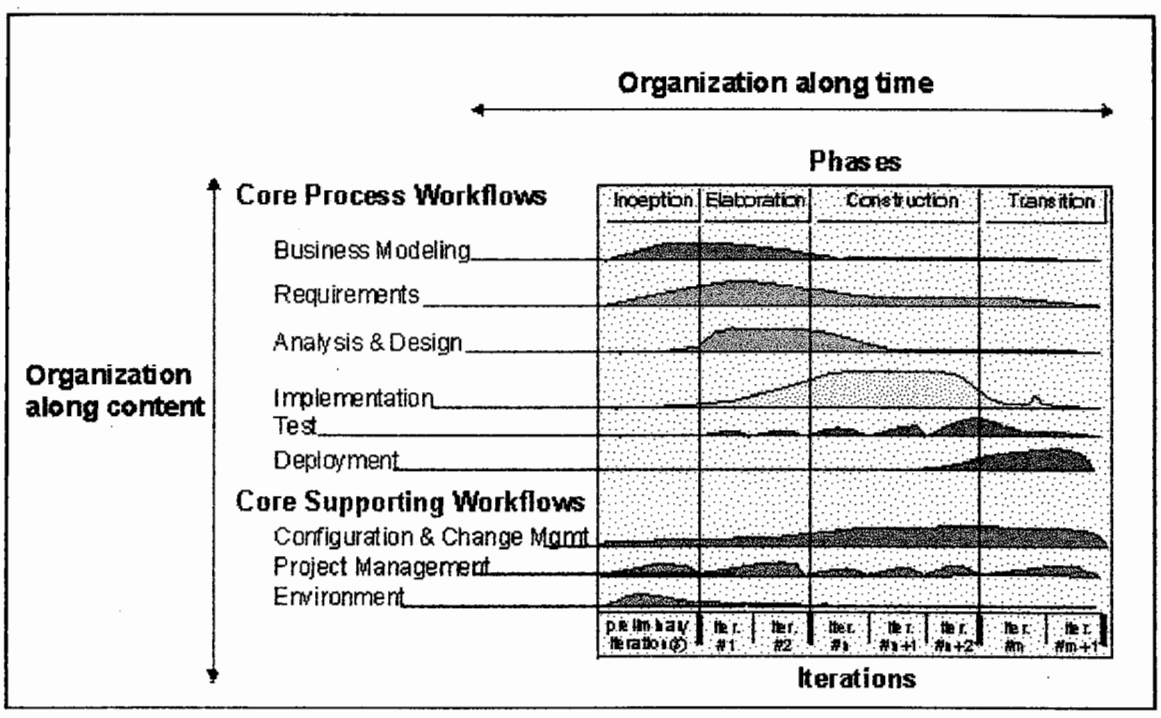


Figure C-1 Rational Unified Process Diagram
(Source: http://www.rational.com/uml)

Organization along Content

The static structure of the Rational Unified Process is represented using four primary modeling elements:

- *Workers*, the "who": the behavior and responsibilities of an individual, or a group of individuals working together as a team.
- *Activities*, the "how": a unit of work that an individual in that role may be asked to perform.

- *Artifacts*, the "what": a piece of information that is produced, modified, or used by a process.
- *Workflows*, the "when": a sequence of activities that produces a result of observable value.

Core Workflows

Rational Unified Process defines nine core workflows, which represent a partitioning of all workers and activities into logical groupings. The core process workflows are divided into:

A. Core "engineering" workflows:
   1. Business modeling workflow
   2. Requirements workflow
   3. Analysis and Design workflow
   4. Implementation workflow
   5. Test workflow
   6. Deployment workflow

B. Three core "supporting" workflow:
   1. Project Management workflow
   2. Configuration and Change Management workflow
   3. Environment workflow.

Unlike in a traditional waterfall model, the process is iterative and these workflows are revisited again and again throughout the lifecyle. The actual complete workflow of a project

interleaves these nine workflows, and repeats them with various emphasis and intensity at each iteration. Figure C-2 illustrates this iterative process.

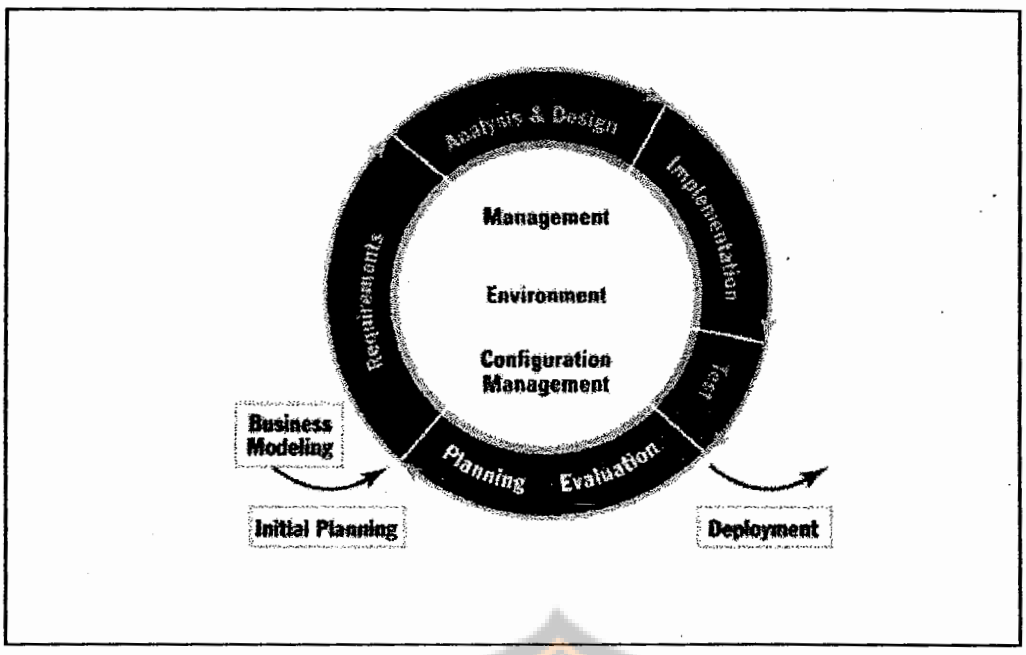


Figure C-2 Rational Unified Process Workflows
(Source: http://www.rational.com/uml)

APPENDIX D: BON STATIC AND DYNAMIC MODEL

Static Model

BON static model contains formal descriptions of class interfaces, their grouping into clusters as well as client and inheritance relations between them, showing the system structure (Waldén and Nerson 1995, 25). It consists of two parts as the following:

1. A collection of very high-level untyped modeling charts
2. A structured description containing fully typed class interfaces and formal specification of software contracts.

The first part can be used to communicate basic ideas to non-technical people, such as end-users and domain experts. Three types of modeling charts are available for this purpose:

1. *System chart*
2. *Cluster chart*
3. *Class chart*

The descriptions of these charts can be found in Appendix E, whereas Appendix G describes their formats.

The second part, a structured description containing fully typed class interfaces and formal specification of software contracts, is the main part of the static model in the BON. To provide users with different view level of the system, three basic levels of static description are available in BON:

- *Class level*: shows class interfaces with their operations and contracts
- *Cluster level*: shows a set of classes
- *System level*: shows a set of clusters.

The system and cluster levels may each comprise several levels of nesting. It is possible to mix the level of detail in a view by showing for example a cluster with only a few class headers expanded into class interfaces. In that case, the description is said to be at the lowest level contained in the view.

The diagram representing possibly nested clusters, class headers, and their relationships is called *static architecture*. This diagram is a bird's eye view of the system, which is zoomable to a more detailed view showing *class interfaces* along with their features and contracts.

Each class interface, as can be seen in Appendix G, consists of a number of sections, some of which may be empty. The sections are:

- *Class header*: consists of the class name, which may be annotated to highlight certain key properties of the class.
- *Indexing clause:* contains general information about the class to be used for browsing and configuration management purposes.
- *Inheritance clause*: lists all parents of the class, if any.
- *Class features*: covers all operational aspects and state aspects of the class operations grouped into two feature clauses, namely public features and restricted features.
- *Class invariant*: contains all assertions about every object of the class.

Static Relation

Classes are grouped into clusters, and clusters may in turn contain both classes and other clusters. Classes and clusters in the static model are related each other in such a relation called *static relation*. There are two kinds of static relations in object-oriented system:

1. *Inheritance relation*
2. *Client/supplier relation* (or *client relation* for short).

Inheritance relation represents a relation in which a class may inherit from other class(es). A client relation between a client class A and a supplier class B means that A uses services supplied by B. There are three types of client relation:

1. *Association*: An association between a client class and a supplier class means that (at system execution time) some instances of the client class may be attached to one or more instances of the supplier class.
2. *Shared association:* A shared association between a client class and a supplier class is a special case meaning that whenever an instance of the client class is attached to an instance of the supplier class, it will always be to the same supplier instance (or one in a fixed set).
3. *Aggregation:* An aggregation relation between a client class and a supplier class means that each client instance may be attached to one or more supplier instances which represent "integral parts" of the client instance.

BON provides several graphical representations as well as textual representations to describe client relation, as well as inheritance relation. See Appendix G for further descriptions.

Dynamic Model

The system's dynamic model specifies system events, what object types are responsible for the creation of other objects, and system execution scenarios representing selected types of system usage with diagrams showing object message passing.

The purpose of BON dynamic model, according to Waldén and Nerson (1995, 90), is to help capture and communicate the idea of how the operations of the high-level classes can fulfil their specifications by calling other operations.

The dynamic model consists of:

1. A set of *dynamic charts*, namely:

- *Event chart*
- *Scenario chart*
- *Object creation chart.*

2. *Dynamic diagram* called *object scenario.*

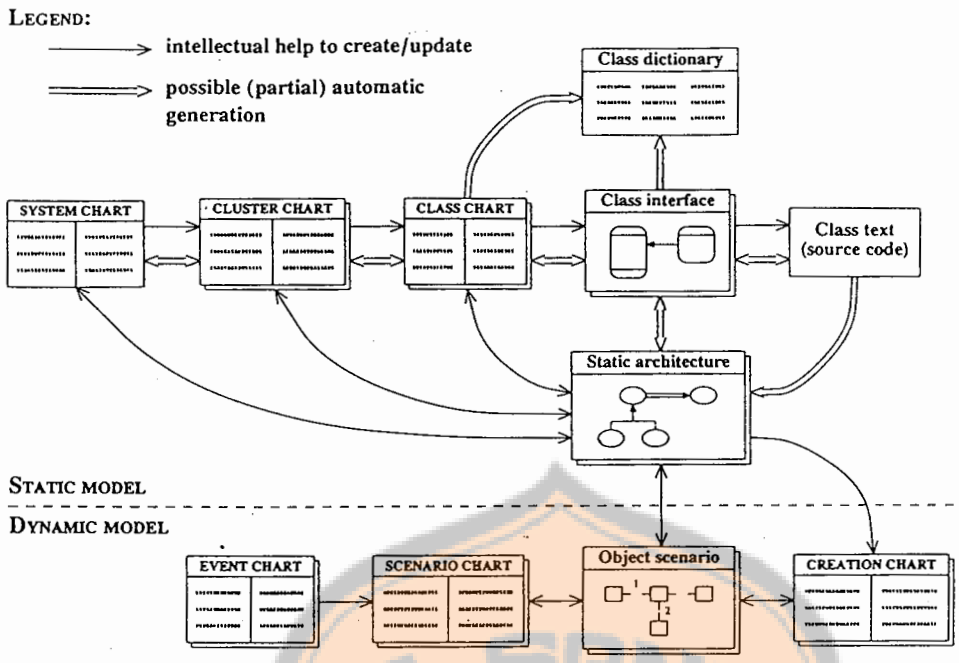
The descriptions of these charts and diagram can be found in Appendix E, whereas Appendix G describes their formats.

APPENDIX E: BON DELIVERABLES

NAME	DESCRIPTION
SYSTEM CHART	• Definition of system and list of associated clusters. • Only one system chart per project. • Subsystems are described through corresponding cluster charts.
CLUSTER CHARTS	• Definition of cluster and lists of associated classes and subclusters, if any. • A cluster may represent a full subsytem or just a group of classes.
CLASS CHARTS	Definition of analysis classes in terms of *commands*, *queries*, and *constraints*.
CLASS DICTIONARY	• Alphabetically sorted list of all classes in the system, showing the cluster of each class and a short description. • Should be generated automatically from the class charts/ interfaces.
STATIC ARCHITECTURE *)	• Set of diagrams representing possibly nested clusters, class headers, and their relationships. • Bird's eye view of the system (zoomable).
CLASS INTERFACES	• Typed definitions of classes with feature signatures and formal contracts. • Detailed view of the system.
CREATION CHARTS	• List of classes in charge of creating instances of other classes. • Usually only one per system, but may be repeated for subsystems if desirable.
EVENT CHARTS	• Set of incoming external events (stimuli) triggering interesting system behavior and set of outgoing external events forming interesting system responses. • May be repeated for subsystems.
SCENARIO CHARTS	• List of object scenarios used to illustrate interesting and representative system behavior. • Subsystems may contain local scenario charts.
OBJECT SCENARIOS	Dynamic diagrams showing relevant object communication for some or all of the scenarios in the scenario chart.

*) Underlined names indicate the most important ones.

APPENDIX F: DEPENDENCIES OF BON DELIVERABLES



APPENDIX G: BON NOTATIONS

BON notation: charts and interfaces

INFORMAL CHARTS

SYSTEM	*SYSTEM_NAME*	Part:
PURPOSE	INDEXING	
Cluster	Description	

EVENTS	*SYSTEM_NAME*	Part:
COMMENT	INDEXING	
External (in/out)	Involved object types	

CLUSTER	*CLUSTER_NAME*	Part:
PURPOSE	INDEXING	
Class/(Cluster)	Description	

SCENARIOS	*SYSTEM_NAME*	Part:
COMMENT	INDEXING	
Scenario 1: Description 1		

CLASS	*CLASS_NAME*	Part:
TYPE OF OBJECT	INDEXING	
Inherits from		
Queries		
Commands		
Constraints		

CREATION	*SYSTEM_NAME*	Part:
COMMENT	INDEXING	
Class	Creates instances of	

CLASS INTERFACE

FEATURE SIGNATURES

*name**, *name*+, *name*++	deferred/effective/redefined
name: *TYPE*	result type
name:{ *TYPE*	aggregation result type
name:(n) *TYPE*	shared result type
{ ^*CLASS_NAME*.*name* }	rename clause
→ *name*: *TYPE*	input argument
?, !	pre- and postcondition

ASSERTION SYMBOLS

Δ *name*	attribute *name* may change
old *expr*	previous value of *expr*
@, ∅	current object, void reference
+, −, *, /, ^, //, \\	arithmetic op
=, ≠, <, ≤, >, ≥	relational op
→, ↔, ¬, **and**, **or**, xor	boolean op
∃, ∀, |, •	predicate logic op
∈, ∉, { }, ..	set op
: *TYPE*	type op

BON notation: static and dynamic diagrams

CLASS HEADERS

reused

persistent

parameterized

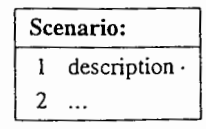
deferred

effective

interfaced

root

SCENARIO BOX



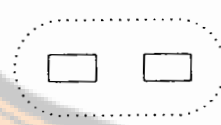
OBJECT HEADERS

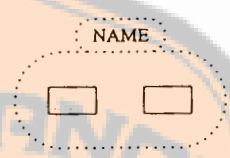
one object

one object (qualified)

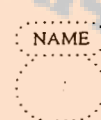
one or more objects

OBJECT GROUPING


object group (unnamed)

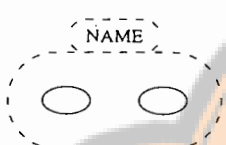


object group



object group (iconized)

CLUSTERING



cluster

cluster (iconized)

STATIC LINKS

inheritance

client association

client aggregation

MULTIPLICITY

of relation

of shared instances

DYNAMIC LINKS

message passing

DYNAMIC LABELS

1, 2 sequence number

STATIC LABELS

name
name1 , *name2*
TYPE [...]
name: *TYPE* [...]
(*name1* , *name2*): *TYPE* [...]
→ *TYPE* [...]

APPENDIX H: BON PROCESS AND STANDARD ACTIVITIES

BON Process

	TASK	DESCRIPTION	GENERAL RESULTS	INPUT SOURCES	BON DELIVERABLES	ACCEPTANCE CRITERIA
GATHERING	1	**Delineate system borderline**	1. Major subsystems 2. User metaphors 3. Incoming and outgoing information flow 4. Major system functionality 5. Typical use cases 6. Reused libraries and reuse policy	1. Customer requirements 2. Domain experts, end-users 3. Knowledge of implementation platforms and run-time environment 4. Comparable systems or prototypes 5. Available reusable components 6. General rules and regulations	1. SYSTEM CHART 2. SCENARIO CHARTS	1. Customer acceptance procedure 2. Quality assurance plan
	2	**List candidate classes**	1. First list of classes based on problem domain terminology 2. Glossary of technical terms and concepts used in problem domain.	1. Problem space 2. User metaphors 3. Specification of implementation platform(s) and external interfaces	CLUSTER CHARTS	Approval by end-users and domain experts
	3	**Select classes and group into clusters**	1. Problem-domain-related classes grouped into clusters 2. First static architecture with some inheritance and client relations	1. Possible subsytems 2. List of candidate classes 3. Glossary of technical terms	1. SYSTEM CHART 2. CLUSTER CHARTS 3. STATIC ARCHITECTURE 4. CLASS DICTIONARY	Class and cluster management system with automatic consistency checks

DESCRIBING	4	**Define classes**	1. Classes defined by **queries, commands,** and **constraints** 2. Updated glossary of terms	1. System and cluster charts 2. Typical use cases 3. Glossary of terms	CLASS CHARTS	End-user/customer acceptance
	5	**Sketch system behaviors**	1. Object creation chart 2. Event and scenario charts 3. Object scenarios	1. Early scenario charts 2. Major system functionality 3. Typical use cases 4. Incoming and outgoing information flow	1. EVENT CHARTS 2. SCENARIO CHARTS 3. CREATION CHARTS 4. OBJECT SCENARIOS	1. Consistency with static model 2. End-user/customer acceptance
	6	**Define public features**	1. Typed class interfaces with public feature signatures (pre- and postconditions, class invariants) 2. Updated static architecture	1. Class charts with queries, commands, and constraints 2. System and cluster charts 3. Glossary of terms	1. CLASS INTERFACES 2. STATIC ARCHITECTURE	1. Consistency with static architecture 2. Features cover external events in event charts and selected scenarios in scenario charts

DESIGNING	7	**Refine system**	1. New design classes 2. Restricted features 3. Detailed network of clusters and classes 4. More complete software contracts	1. Class interfaces 2. Static architecture	1. CLASS INTERFACES 2. STATIC ARCHITECTURE 3. EVENT CHARTS 4. OBJECT SCENARIOS 5. CLASS DICTIONARY	1. Consistency with object creation chart 2. Consistency with object scenarios
	8	**Generalize**	1. Updated class interfaces and static architecture 2. More deferred classes	Static model	1. CLASS INTERFACES 2. STATIC ARCHITECTURE 3. CLASS DICTIONARY	Consistency with object scenarios
	9	**Complete and review system**	1. Reviewed static and dynamic models 2. Reusable class interfaces 3. Class text templates	1. Static and dynamic models 2. Creation and event charts	1. CLASS INTERFACES 2. STATIC ARCHITECTURE 3. EVENT CHARTS 4. OBJECT SCENARIOS 5. CLASS DICTIONARY	1. Syntactic validation of each class 2. Consistency of inherited and combined class invariants 3. Consistency of pre- and post-conditions of polymorphic routines

BON Standard Activities

1. Finding classes
2. Classifying
3. Clustering
4. Defining class features
5. Selecting and describing object scenarios
6. Working out contracting conditions
7. Assessing reuse
8. Indexing and documenting
9. Evolving the system architecture

APPENDIX I: DESCRIPTION OF THE CASE STUDY

BOOK CIRCULATION SYSTEM

In this system, faculty members and students borrow and return books. Only students get fined for books returned past the due date. Seven use cases/scenario are supported, the first three of which have the librarian as actor.

Use Case 1: Facilitate Borrow

The librarian has been given the borrower's ID card and the actual book. She first enters the ID number into the system and then the system validates this number and displays whether or not the ID number is valid and whether or not the borrower may borrow books (she may not if she has overdue books or if she has reached her borrowing limit). If the borrower is allowed to borrow a book, the librarian then enters the book catalog number. If the book exists, and is available for borrowing, the system displays that the book has been recorded as borrowed.

Use case 2: Facilitate Return

The librarian has been given the actual book. She enters the book catalog number. If the book is valid, the system displays that the book has been recorded as returned and prints overdue information, if applicable.

Use case 3: Facilitate Overdue Payment

The librarian enters the ID number of a student. The system then displays an overdue statement for this student, if applicable. The librarian then confirms with the system that the student has paid the corresponding amount and the system displays that the student's record has been updated to reflect the paid fine.

Use case 4: Search Item

The borrower enters search information (title, subject, or author). The system then lists all books that apply. The system then prints out all pertinent information (catalog number, whether or not the book is available, etc.)

Use case 5: Hold Item

A borrower may cause a book to be held for a full day, signifying her intention to borrow the book. This is done by simply pressing some key after she has located the book using scenario to search book described above. The borrower will be prompted for a name and a password before performing this transaction.

Use case 6: Reserve Item

Using a process similar to the process of intention to borrow books, a faculty member may place a book on reserve for a specified duration. Books on reserve may not be borrowed. A name and a password are also required before performing this transaction.

Use case 7: Query on Transaction

After giving a name and a password, a borrower may find out from the system which books she has borrowed and when these are due.

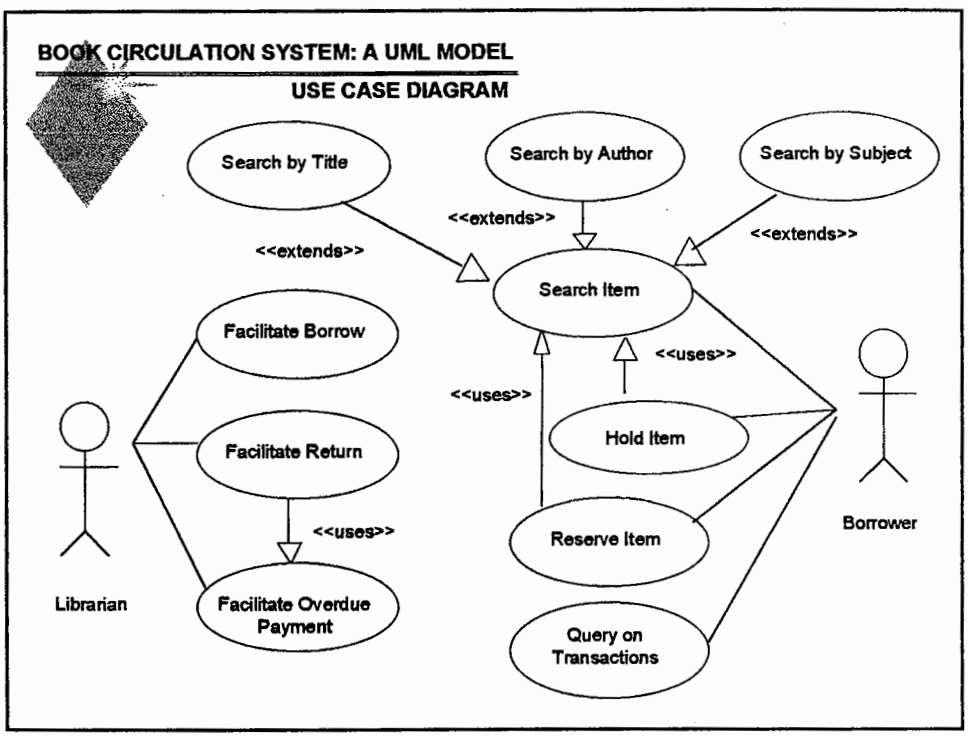
APPENDIX J

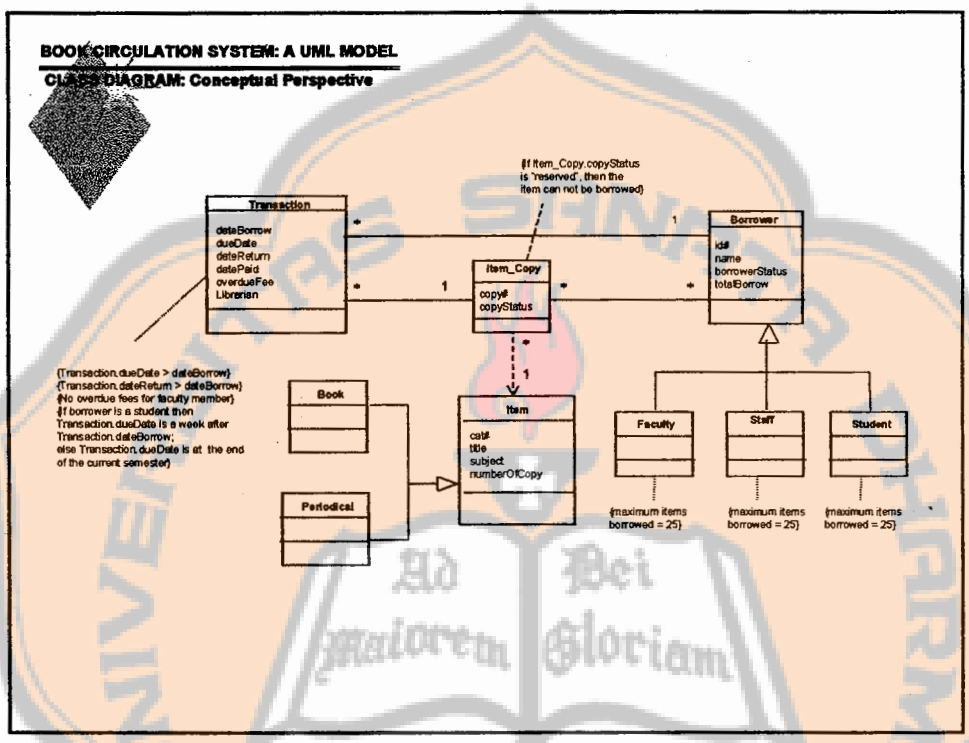
BOOK CIRCULATION SYSTEM :

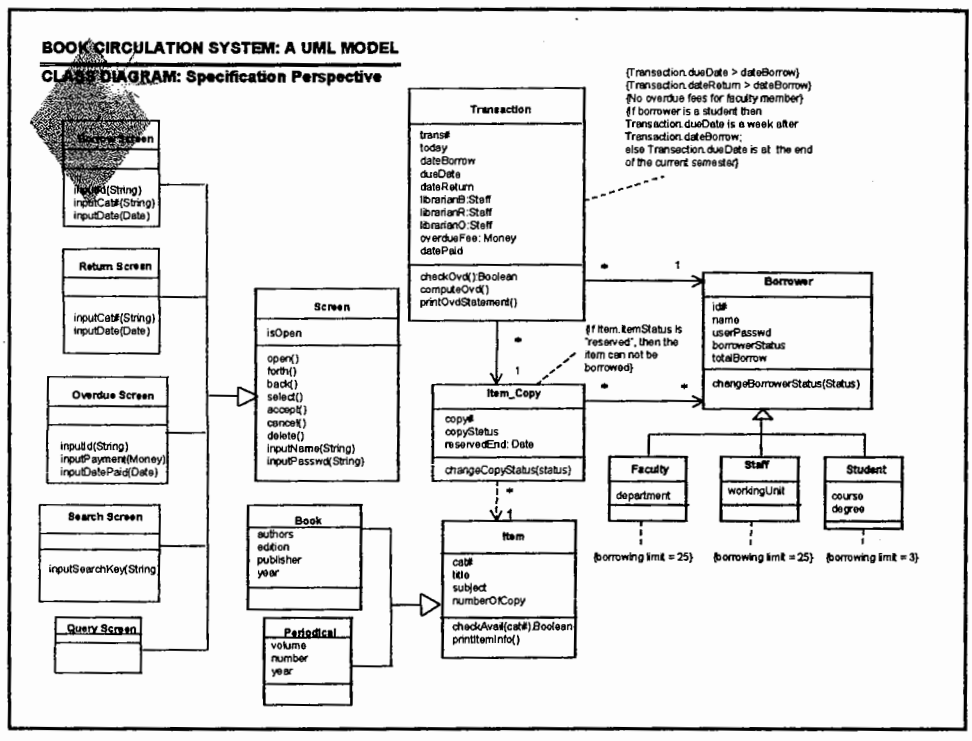
A UML MODEL

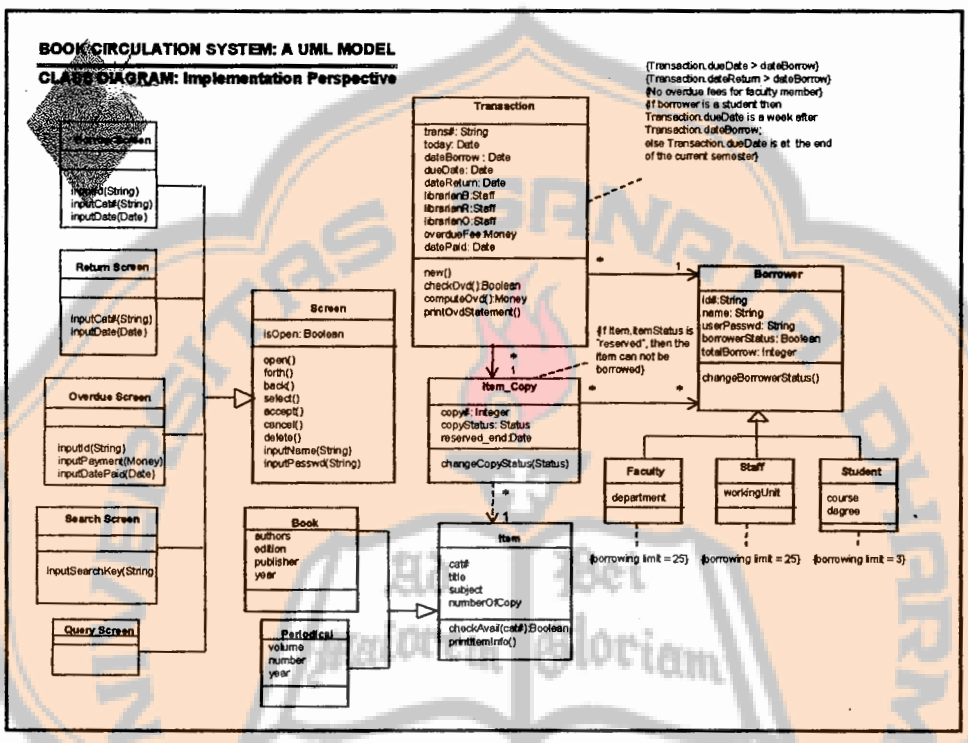
1. USE CASE DIAGRAM
2. CLASS DIAGRAM: Conceptual Perspective
3. CLASS DIAGRAM: Specification Perspective
4. CLASS DIAGRAM: Implementation Perspective
5. COLLABORATION DIAGRAM
6. ACTIVITY DIAGRAM: Conceptual Perspective

Author : Paulina H. Prima Rosa
Last updated : Dec 16,1998
C:/data/thesis/model-2.ppt





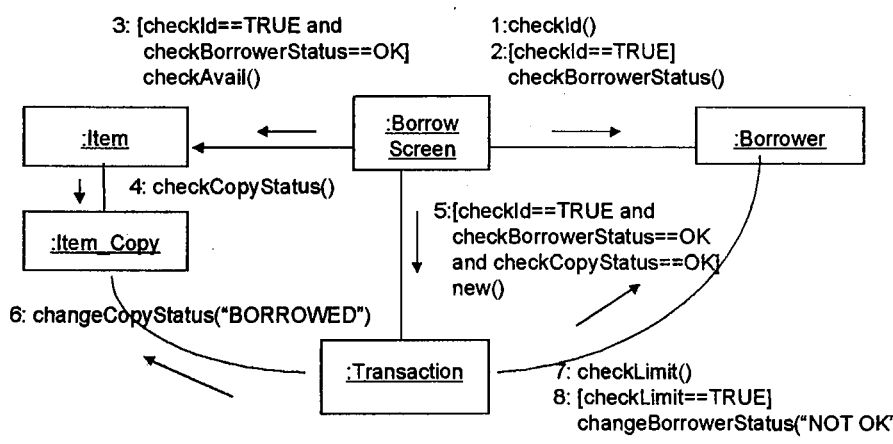




BOOK CIRCULATION SYSTEM: A UML MODEL

COLLABORATION DIAGRAM

SCENARIO 1: FACILITATE BORROW



BOOK CIRCULATION SYSTEM: A UML MODEL

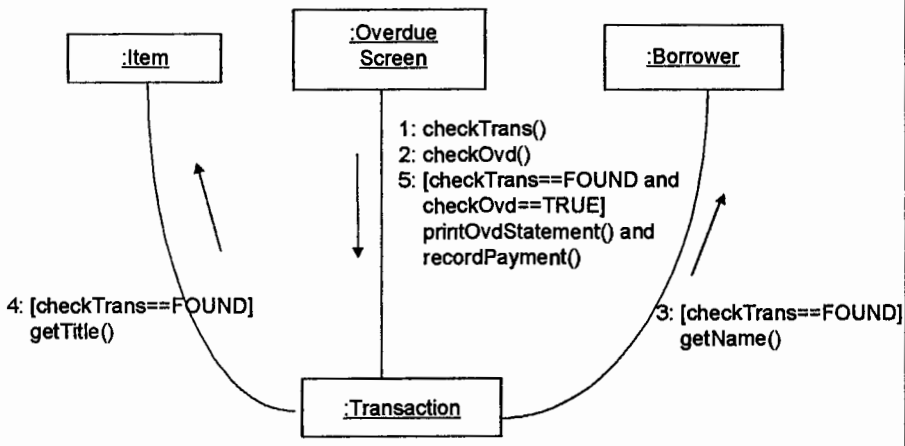
COLLABORATION DIAGRAM

SCENARIO 2: FACILITATE RETURN

BOOK CIRCULATION SYSTEM: A UML MODEL

COLLABORATION DIAGRAM

SCENARIO 3: FACILITATE OVERDUE PAYMENT



BOOK CIRCULATION SYSTEM: A UML MODEL

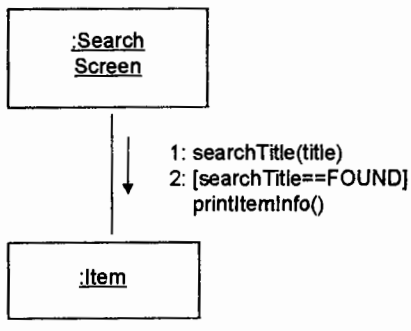
COLLABORATION DIAGRAM

SCENARIO 4a: SEARCH ITEM (by Catalog Number)

BOOK CIRCULATION SYSTEM: A UML MODEL

COLLABORATION DIAGRAM

SCENARIO 4b: SEARCH ITEM (by Title)



BOOK CIRCULATION SYSTEM: A UML MODEL

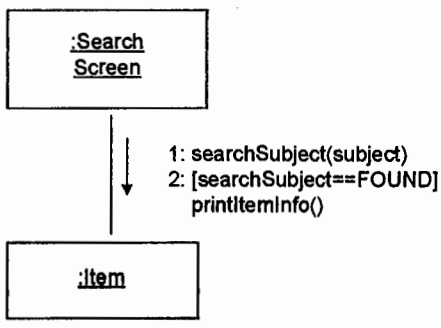
COLLABORATION DIAGRAM

SCENARIO 4c: SEARCH ITEM (by Authors)

BOOK CIRCULATION SYSTEM: A UML MODEL

COLLABORATION DIAGRAM

SCENARIO 4c: SEARCH ITEM (by Subject)



BOOK CIRCULATION SYSTEM: A UML MODEL

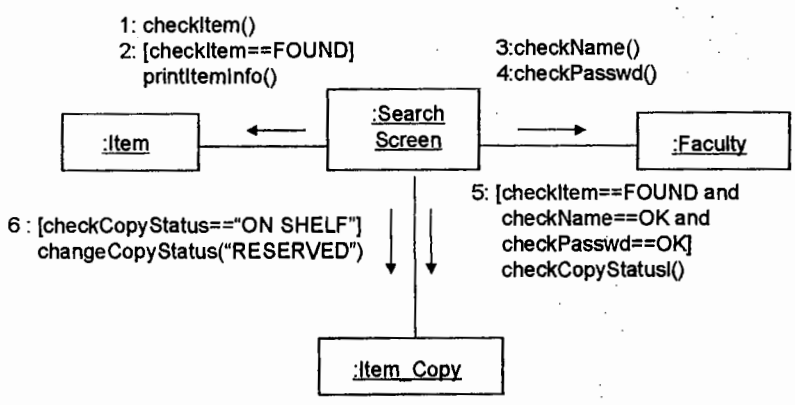
COLLABORATION DIAGRAM

SCENARIO 5: HOLD ITEM

BOOK CIRCULATION SYSTEM: A UML MODEL

COLLABORATION DIAGRAM

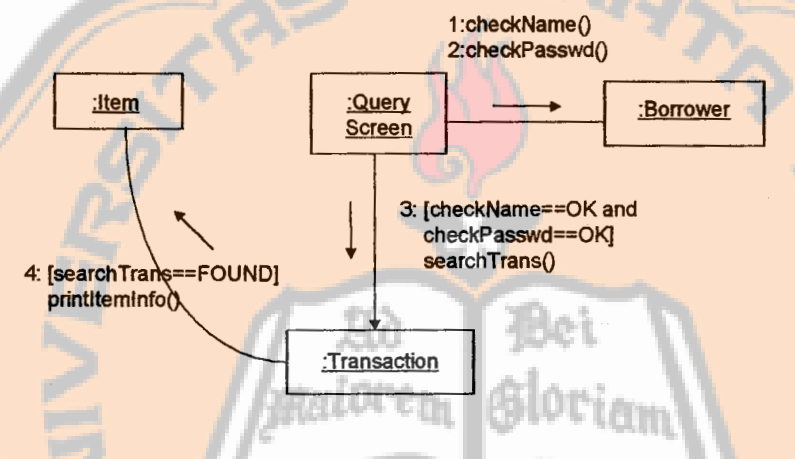
SCENARIO 6: RESERVE ITEM

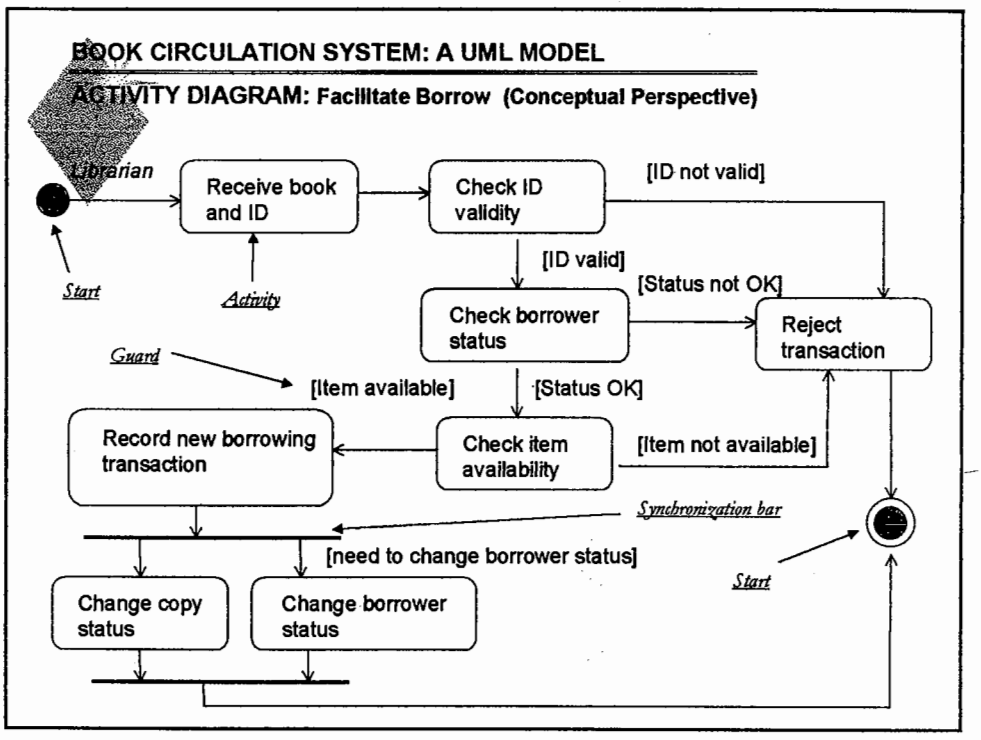


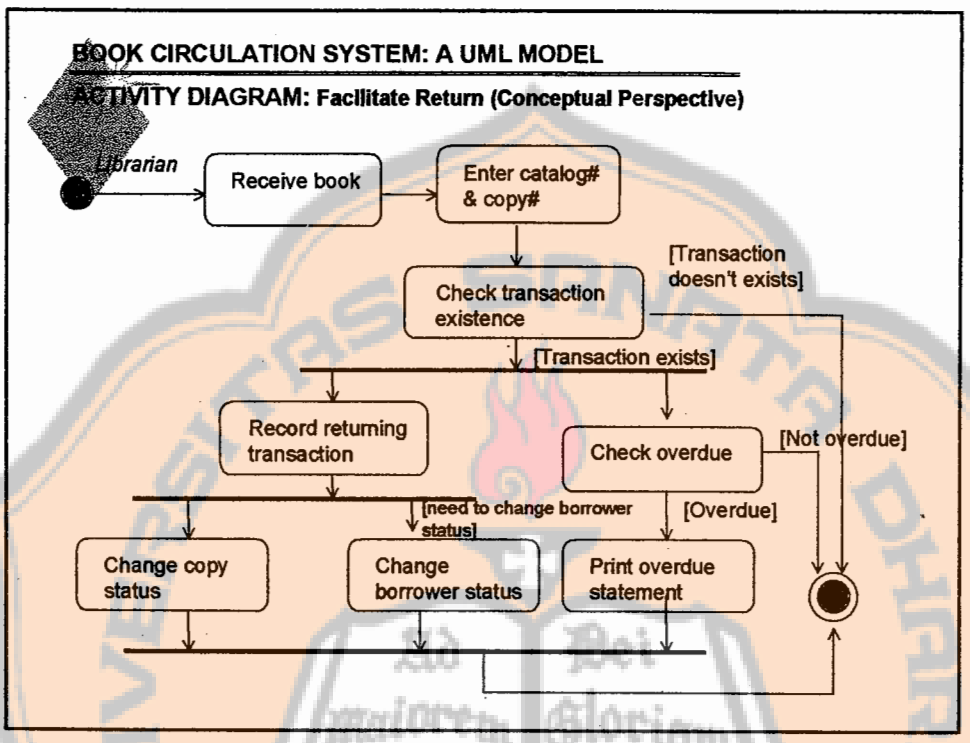
BOOK CIRCULATION SYSTEM: A UML MODEL

COLLABORATION DIAGRAM

SCENARIO 7: QUERY ON TRANSACTIONS

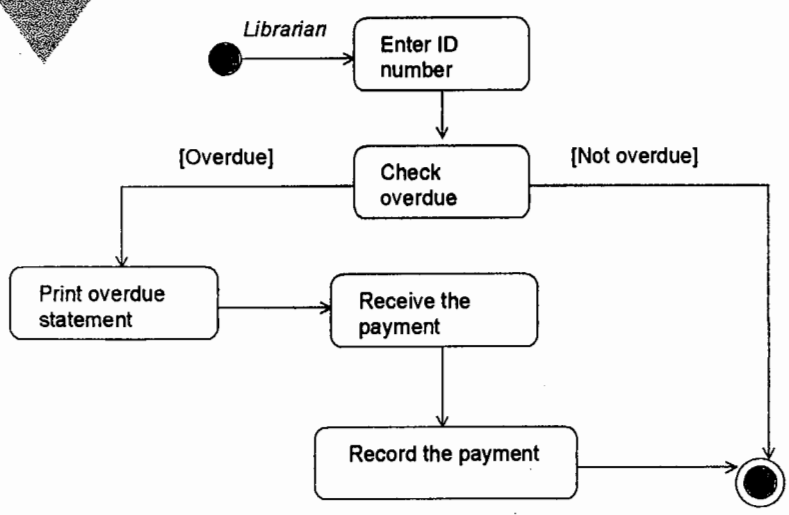






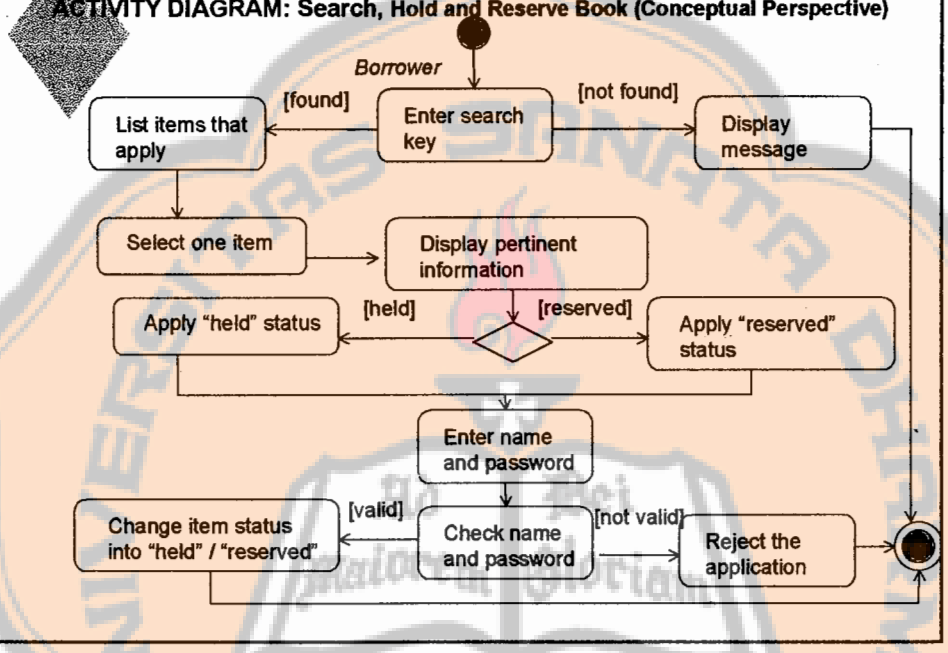
BOOK CIRCULATION SYSTEM: A UML MODEL

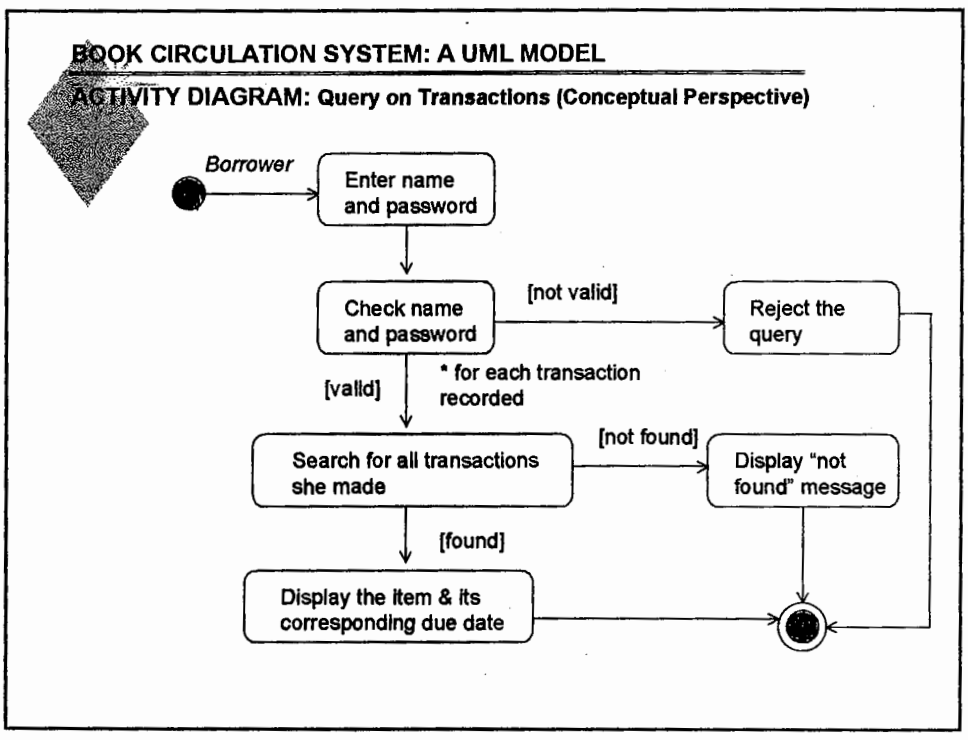
ACTIVITY DIAGRAM: Facilitate Overdue Payment (Conceptual Perspective)



BOOK CIRCULATION SYSTEM: A UML MODEL

ACTIVITY DIAGRAM: Search, Hold and Reserve Book (Conceptual Perspective)





APPENDIX K

BOOK CIRCULATION SYSTEM :

A BON MODEL

1. FIRST CANDIDATE CLASSES
2. CLUSTER CHARTS
3. STATIC ARCHITECTURE
4. CLASS CHARTS
5. EVENT CHARTS
6. SCENARIO CHARTS
7. OBJECT CREATION CHARTS
8. OBJECT SCENARIOS
9. FORMAL CLASS DESCRIPTION

Author : Paulina H. Prima Rosa
Last updated : Dec 17,1998
C:/data/thesis/model-2.doc
C:/data/thesis/model-2.ppt

1. FIRST CANDIDATE CLASSES

From the description of the case, some candidate classes are identified.

CLUSTER	*BOOK CIRCULATION SYSTEM*	Part: 1/1
PURPOSE System to facilitate borrowing and returning books, facilitate overdue payments, and keep track of book circulation in a library.		**INDEXING** **keywords:** book circulation system, first gathered analysis classes

Class/(Clusters)	Description
BOOK	Book available in the library
BOOK_CIRCULATION	System for managing book circulation in a library
BORROW_SCREEN	Menu option and data entry to facilitate borrow
FACULTY	Person who teaches in the school where the library is located
LIBRARIAN	Person who facilitate circulation in the library
OVERDUE_SCREEN	Menu option and data entry to facilitate overdue payment
PERIODICAL	Periodical available in the library
QUERY_SCREEN	Menu option to facilitate query on transactions
RETURN_SCREEN	Menu option and data entry to facilitate return
SEARCH_SCREEN	Menu option and data entry to facilitate item searching for borrowers
STAFF	Person who works as administrative staff in the school where the library is located
STUDENT	Person who studies in the school where the library is located
TRANSACTION	Record of every transaction made (borrow, return, overdue payment)

2. CLUSTER CHART

From the first identified set of classes, we select classes from the candidates, then classify and grouped some of them into three cluster structures.

CLUSTER	*BOOK CIRCULATION SYSTEM*	**Part:** 1/1
PURPOSE System to facilitate borrowing and returning books, facilitate overdue payments, and keep track of book circulation in a library.		**INDEXING** **keywords:** book circulation system, selected and grouped classes

Class/(Clusters)	Description
BOOK_CIRCULATION	Root class containing system for managing book circulation in a library
BORROWER	Deferred class containing persons who are allowed to borrow item
ITEM	Deferred class containing items available to borrow
ITEM_COPY	Class containing copies of items available
SCREEN	Deferred class containing options and data entry
TRANSACTION	Record of every transaction made (borrow, return, overdue payment)
(BORROWERS)	Subcluster containing classes inheriting from class *BORROWER*
(ITEMS)	Subcluster containing classes inheriting from class ITEM
(SCREENS)	Subcluster containing classes inheriting from class SCREEN

CLUSTER	*BORROWERS*	**Part:** 1/1
PURPOSE Different types of borrower classes		**INDEXING** **keywords:** book circulation system, borrower

Class/(Clusters)	Description
FACULTY	Person who teaches in the school where the library is located
STAFF	Person who works as administrative staff in the school where the library is located (including librarians)
STUDENT	Person who studies in the school where the library is located

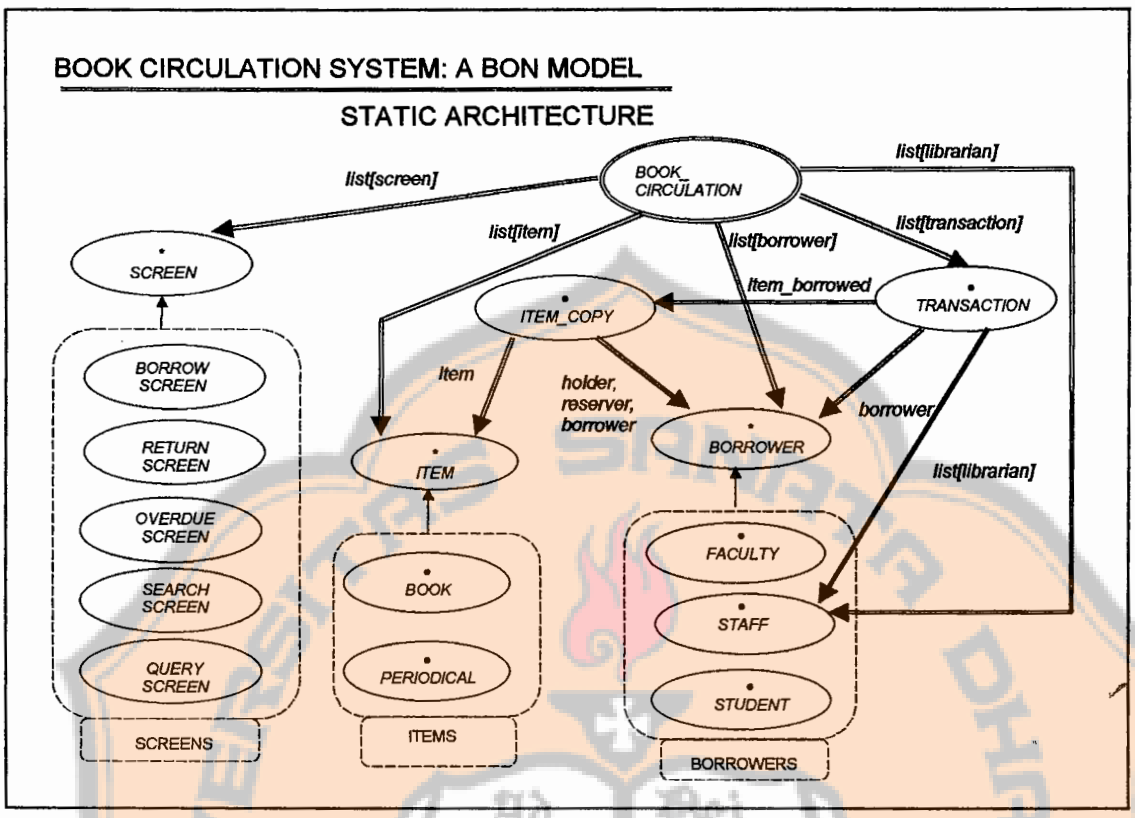
CLUSTER	*ITEMS*	**Part:** 1/1
PURPOSE Different types of items available to borrow		**INDEXING** **keywords:** book circulation system, item available

Class/(Clusters)	Description
BOOK	Book available in the library
PERIODICAL	Periodical available in the library

CLUSTER	SCREENS	Part: 1/1
PURPOSE Different types of screen classes encapsulating user options		INDEXING **Keywords:** book circulation system, user screen

Class/(Clusters)	Description
BORROW_SCREEN	Menu option and data entry to facilitate borrow
SEARCH_SCREEN	Menu option and data entry to facilitate item searching
OVERDUE_SCREN	Menu option and data entry to facilitate overdue payment
RETURN_SCREEN	Menu option and data entry to facilitate return
QUERY_SCREEN	Menu option to facilitate query on transactions

3. STATIC ARCHITECTURE



4. CLASS CHARTS

Having selected and grouped the analysis classes, we proceed to look at their operations. The following are the class charts of all the classes defined above.

CLASS	*BOOK_CIRCULATION*	**Part:** 1/1
TYPE OF OBJECT Generic book circulation system in a library	**INDEXING** **cluster:** *BOOK_CIRCULATION_SYSTEM* **keywords:** book circulation system	
Queries	Library's name. School. Menu. Collection. Member. Librarian	
Commands	Open. Close. Check password.	
Constraints	Only authorized person may initiate the system.	

CLASS	*BORROWER*	**Part:** 1/1
TYPE OF OBJECT Person who is allowed to borrow items	**INDEXING** **Cluster:** *BOOK_CIRCULATION_SYSTEM* **Keywords:** book circulation system, borrower	
Queries	ID number. Name. Password. Total borrow. Borrowing limit. Borrower status (allowed/not allowed to borrow).	
Commands	Change status.	
Constraints	The ID number must be unique. Borrower is not allowed to borrow if she has overdue books or if she has reached borrowing limit.	

CLASS	*FACULTY*	**Part:** 1/1
TYPE OF OBJECT Faculty member who is allowed to borrow items	**INDEXING** **cluster:** *BORROWERS* **keywords:** faculty	
Inherits from	*BORROWER*	
Queries	Department	
Constraints	Borrowing limit = 25	

CLASS	*STAFF*	**Part:** 1/1
TYPE OF OBJECT Staff member who is allowed to borrow items	**INDEXING** **cluster:** *BORROWERS* **keywords:** staff	
Inherits from	*BORROWER*	
Queries	Working unit	
Constraints	Borrowing limit = 25	

CLASS	*STUDENT*	Part: 1/1
TYPE OF OBJECT Student who is allowed to borrow items	**INDEXING** **cluster:** *BORROWERS* **keywords:** student	
Inherits from	*BORROWER*	
Queries	Course, Degree	
Constraints	Borrowing limit = 3	

CLASS	*ITEM*	Part: 1/1
TYPE OF OBJECT Item available to borrow	**INDEXING** **cluster:** *BOOK_CIRCULATION_SYSTEM* **keywords:** book circulation system, item	
Queries	Catalog number. Title. Subject. Number of copy.	
Commands	Check item availability. Print item information.	
Constraints	Catalog number must be unique.	

CLASS	*BOOK*	Part: 1/1
TYPE OF OBJECT Book available to borrow	**INDEXING** **cluster:** *ITEMS* **keywords:** book available	
Inherits from	*ITEM*	
Queries	Authors. Edition. Publishers. Year.	
Commands	Search item by authors	

CLASS	*PERIODICAL*	Part: 1/1
TYPE OF OBJECT Periodical available in the library	**INDEXING** **Cluster:** *ITEMS* **Keywords:** periodical available	
Inherits from	*ITEM*	
Queries	Volume. Number. Year.	

CLASS	*ITEM_COPY*	Part: 1/1
TYPE OF OBJECT Copy of item available		**INDEXING** **cluster:** *BOOK_CIRCULATION_SYSTEM* **keywords:** book circulation system, item copy
Queries	Catalog number. Copy number. Copy status. Holder. Reserver. Borrower. Reserved end.	
Commands	Change copy status (on shelf/borrowed/held/reserved).	
Constraints	Item with 'held' status could only be borrowed by the borrower who placed the 'held' status. Only faculty member may change item status into reserved. Item with 'reserved' status could not be borrowed for a specified duration.	

CLASS	*SCREEN*	Part: 1/1
TYPE OF OBJECT Option screen and data entry		**INDEXING** **Cluster:** *BOOK_CIRCULATION_SYSTEM* **Keywords:** option screen
Queries	Is this screen open?	
Commands	Input user name. Input password. Open this menu. Shift focus to next entry. Shift focus to previous entry. Select. Accept. Cancel. Delete.	
Constraints	Only person with a correct password may enter to this menu.	

CLASS	*BORROW_SCREEN*	Part: 1/1
TYPE OF OBJECT Option screen and data entry to facilitate borrow		**INDEXING** **cluster:** *SCREEN* **keywords:** option screen, borrow
Inherits from	*SCREEN*	
Commands	Input ID number. Input catalog number. Input date borrow.	

CLASS	*RETURN_SCREEN*	Part: 1/1
TYPE OF OBJECT Option screen and data entry to facilitate return		**INDEXING** **cluster:** *SCREEN* **keywords:** option screen, return
Inherits from	*SCREEN*	
Commands	Input catalog number. Input date return.	

CLASS	*OVERDUE_SCREEN*	Part: 1/1
TYPE OF OBJECT Option screen and data entry to facilitate overdue payment	**INDEXING** **cluster:** *SCREEN* **keywords:** option screen, overdue payment	
Inherits from	*SCREEN*	
Commands	Input ID number. Input overdue payment. Input payment date.	

CLASS	*SEARCH_SCREEN*	Part: 1/1
TYPE OF OBJECT Option screen and data entry to facilitate item searching	**INDEXING** **cluster:** *SCREEN* **keywords:** borrower, item searching	
Inherits from	*SCREEN*	
Commands	Input search key.	

CLASS	*QUERY_SCREEN*	Part: 1/1
TYPE OF OBJECT Option screen and data entry to facilitate query on transactions	**INDEXING** **Cluster:** *SCREEN* **keywords:** borrower, query on transactions	
Inherits from	*SCREEN*	

CLASS	*TRANSACTION*	Part: 1/1
TYPE OF OBJECT Record of every transaction made	**INDEXING** **cluster:** *BOOK_CIRCULATION_SYSTEM* **keywords:** transaction, borrow, return, overdue payment	
Queries	Transaction number. Item. Borrower. Today's date. Date borrow. Due date. Date return. Date paid. Librarian who facilitates borrowing. Librarian who facilitates returning. Librarian who facilitates overdue payment. Is the item overdue? Overdue fees.	
Commands	Acknowledge new borrowing transaction. Record date return. Compute overdue fees. Print overdue statement. Record overdue payment.	
Constraints	Only person with valid ID may borrow items. Only items with valid catalog number may be borrowed. Only items with valid catalog number may be returned. Date return should be after or the same as date borrow. For students, items are due a week after it is borrowed. For faculty and staff members, items are due at the end of the current semester. Only students and staff members could be assessed overdue fees.	

5. EVENT CHARTS

Using the description of the case, we can identify a set of significant incoming external events that will lead to interesting system behavior. These events result in the creation of new objects, or the passing of information between active objects, or the entry and propagation of external data into active objects. Some of them will also cause significant outgoing system responses to be produced by internal actions.

EVENT	*BOOK CIRCULATION SYSTEM*	**Part:** 1/2
COMMENT Typical incoming events triggering interesting behavior.		**INDEXING** **keywords:** external events, book circulation

External (incoming)	**Involved object types**
A menu option is activated	*BOOK_CIRCULATION, SCREENS**
An ID is validated	*BORROWER**
A catalog number is validated	*ITEM**
A borrowing request is accepted or rejected	*BORROW_SCREEN, BORROWER*, ITEM*, ITEM_COPY, TRANSACTION*
An accepted borrowing is recorded	*BORROW_SCREEN, BORROWER*, ITEM_COPY, TRANSACTION*
An item is returned	*BORROWER, ITEM_COPY, RETURN_SCREEN, TRANSACTION*
Overdue statement is issued	*ITEM*, OVERDUE_SCREEN, STAFF, STUDENT, TRANSACTION*
Overdue fees is paid	*OVERDUE_SCREEN, STAFF, STUDENT, TRANSACTION*
An item searching is performed	*BORROWER*, SEARCH_SCREEN, ITEM**
A reservation is specified	*SEARCH_SCREEN, FACULTY, ITEM*, ITEM_COPY*
A holding is specified	*BORROWER*, ITEM*, ITEM_COPY*
A query on transactions is performed	*BORROWER*, QUERY_SCREEN, ITEM*, TRANSACTION*

Another chart captures outgoing events that are system responses triggered after a chain of external incoming and internal events is activated.

EVENT	*BOOK CIRCULATION SYSTEM*	**Part:** 2/2
COMMENT Typical outgoing events triggering interesting behavior.		**INDEXING** **keywords:** internal events, book circulation

Internal (outgoing)	**Involved object types**
Overdue information is printed	*OVERDUE_SCREEN, STUDENT, STAFF, TRANSACTION*
Overdue statement is printed	*OVERDUE_SCREEN, STUDENT, STAFF, TRANSACTION*
All pertinent information corresponding to a searched item is displayed	*SEARCH_SCREEN, ITEM**
All transactions made by a particular borrower are displayed	*BORROWER*, QUERY_SCREEN, TRANSACTION*

6. SCENARIO CHART

SCENARIOS	*BOOK CIRCULATION SYSTEM*	Part: 1/1
COMMENT Typical behaviors triggered by users	**INDEXING** **Keywords:** book circulation, borrow, held, overdue, query, return, reserve, search, transaction.	
Facilitate borrow: The librarian has been given the borrower's ID card and the actual book. She first enters the ID number into the system and then the system validates this number and displays whether or not the ID number is valid and whether or not the borrower may borrow books (she may not if she has overdue books or if she has reached her borrowing limit). If the borrower is allowed to borrow a book, the librarian then enters the book catalog number. If the book exists and is available for borrowing, the system displays that the book has been recorded as borrowed.		
Facilitate return: The librarian has been given the actual book. She enters the book catalog number. If the book is valid, the system displays that the book has been recorded as returned and prints overdue information, if applicable.		
Facilitate overdue payment: The librarian enters the ID number of a student. The system then displays an overdue statement for this student, if applicable. The librarian then confirms with the system that the student has paid the corresponding amount and the system displays that the student's record has been updated to reflect the paid fine.		
Search item: The borrower enters search information (title, subject, or author). The system then list all books that apply. The system then prints out all pertinent information (catalog number, whether or not the book is available, etc.)		
Hold item: A borrower may cause a book to be held for a full day, signifying her intention to borrow the book. This is done by simply pressing some key after she has located the book using scenario to search book described above. The borrower will be prompted for a name and a password before performing this transaction.		
Reserve item: Using a process similar to the process of intention to borrow books, a faculty member may place a book on reserve for a specified duration. Books on reserve may not be borrowed. A name and a password are also required before performing this transaction.		
Query on transaction: After giving a name and a password, a borrower may find out from the system which books she has borrowed and when these are due.		

7. CREATION CHARTS

To keep track the creation of objects in high-level analysis, an object creation chart is produced. This chart lists all classes along with instances of classes they may create.

CREATION	*BOOK CIRCULATION SYSTEM*	**Part:** 1/1
COMMENT List of classes creating objects in the system.		**INDEXING** **Keywords: object** creation, book circulation

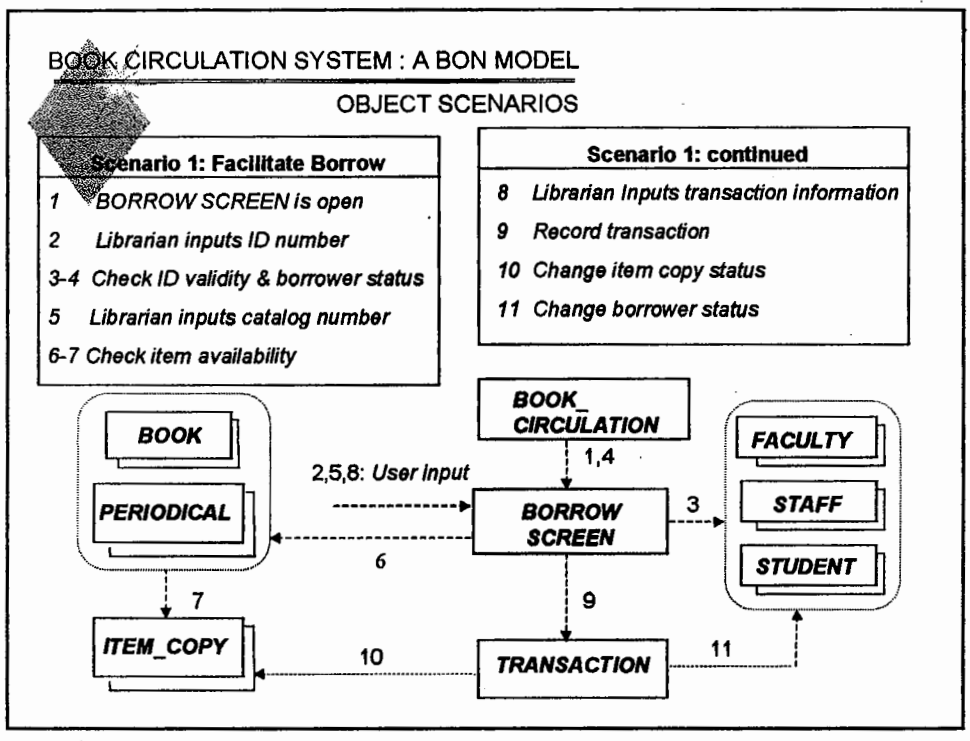
Class	**Creates instances of**
BOOK_CIRCULATION	*BORROWER, BOOK, PERIODICAL, BORROW SCREEN, RETURN SCREEN, OVERDUE SCREEN, SEARCH SCREEN, QUERY SCREEN*
*BORROWER**	*TRANSACTION*
*ITEM**	*ITEM_COPY, TRANSACTION*

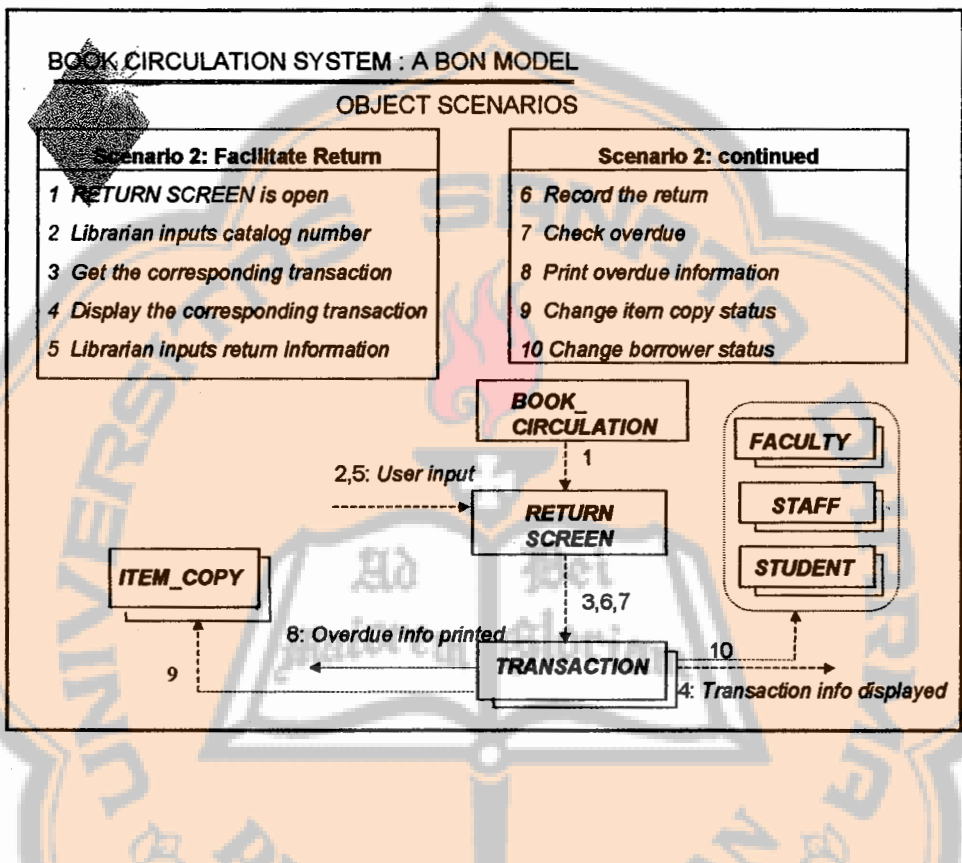
8. OBJECT SCENARIOS

For every scenario described in the scenario chart, we can depict dynamic diagrams called object scenarios showing relevant object communication. See the next pages.

9. FORMAL CLASS DESCRIPTION

Based on the class charts, we can construct more formal descriptions of classes. These description will then serve as a software contract for the implementation phase. See the next pages for the formal class descriptions.

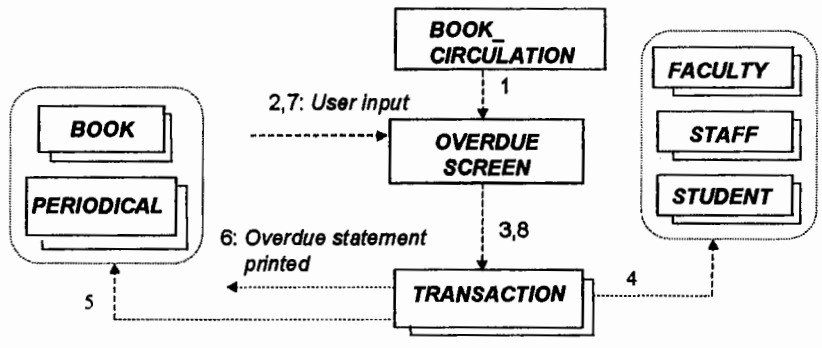




BOOK CIRCULATION SYSTEM : A BON MODEL

OBJECT SCENARIOS

Scenario 3: Facilitate Overdue Payment	Scenario 3: continued
1 OVERDUE SCREEN is open	*4-6 Issue overdue statement*
2 Librarian inputs ID number	*7 Librarian inputs payment information*
3 Get the corresponding transaction	*8 Record the payment*



BOOK CIRCULATION SYSTEM : A BON MODEL

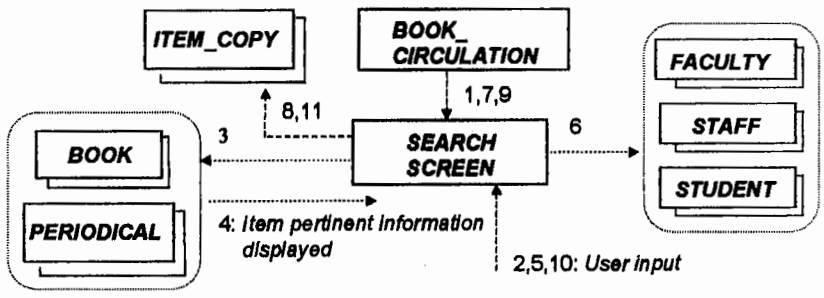
OBJECT SCENARIOS

Scenario 4: Search Item
1 SEARCH SCREEN is open
2 Borrower inputs search key
3 Get the corresponding items
4 Display item pertinent information

BOOK CIRCULATION SYSTEM : A BON MODEL

OBJECT SCENARIOS

Scenario 5: Hold Item	Scenario 5: continued
1 SEARCH SCREEN is open	5 Borrower enters name and password
2 Borrower inputs search key	6-7 Verify name and password
3 Get the corresponding items	8-9 Check item status
4 Display item pertinent information	10-11 Change item status into "held"



BOOK CIRCULATION SYSTEM : A BON MODEL

OBJECT SCENARIOS

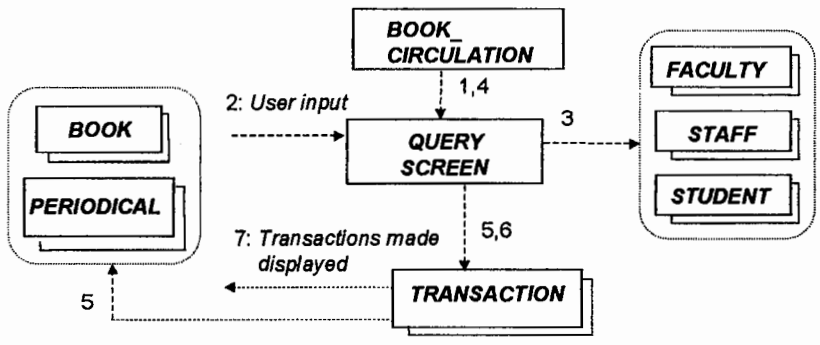
Scenario 6: Reserve Item	Scenario 6: continued
1 SEARCH SCREEN is open	5 Borrower enters name and password
2 Borrower inputs search key	6-7 Verify name and password
3 Get the corresponding items	8-9 Check item status
4 Display item pertinent information	10-11 Change item status into "reserved"

BOOK CIRCULATION SYSTEM : A BON MODEL

OBJECT SCENARIOS

Scenario 7: Query on Transactions
1 QUERY SCREEN is open
2 Borrower inputs name & password
3-4 Verify name and password

Scenario 7: continued
5-6 Search for all transactions she made
7 Display all items she borrows with the corresponding due dates



BOOK CIRCULATION SYSTEM : A BON MODEL

FORMAL CLASS DESCRIPTION

FACULTY + •
Inherits: *BORROWER*
department: VALUE
Invariant
borrowing_limit = 25

STAFF + •
Inherits: *BORROWER*
working_unit: VALUE
Invariant
borrowing_limit = 25

STUDENT + •
Inherits: *BORROWER*
course, degree: VALUE
Invariant
borrowing_limit = 3

BORROWER *
id#, name : VALUE
usr_password: VALUE
– User password
borrower_status: BOOLEAN
– Allowed or not allowed to borrow
total_borrow: INTEGER
– Number of items borrowed
change_status
– Change borrower status
→ *total_borrow: INTEGER; overdue:BOOLEAN*
! *borrower_status ↔ total_borrow < borrowing_limit and overdue=∅*
Invariant
∀ p,q: BORROWER | p≠q • p.id_number ≠ q.id_number

BOOK CIRCULATION SYSTEM : A BON MODEL

FORMAL CLASS DESCRIPTION

BOOK + •

Inherits: *ITEM*

authors, edition, publishers, year: VALUE

PERIODICAL + •

Inherits: *ITEM*

volume, number, year: VALUE

ITEM *

cat#, title, subject: VALUE
number_of_copy:INTEGER
-- Number of copy of the item

check_avail:BOOLEAN
-- Check item availability
→ *cat_number:VALUE*

print_item_info
-- Print item pertinent information
→ *cat_number: VALUE*

Invariant

∀ *p,q: ITEM | p≠q • p.item_code ≠ q.item_code*

BOOK CIRCULATION SYSTEM : A BON MODEL

FORMAL CLASS DESCRIPTION

ITEM_COPY •

cat#: VALUE
copy_number:INTEGER

copy_status:VALUE
-- On shelf/borrowed/held/reserved

change_copy_status
-- Change item copy status
→ *new_status: STATUS*
? *new_status="reserved" • reserver:FACULTY;*
copy_status="held" **and** *new_status="borrowed" •*
borrower=holder
! *copy_status=new_status*

holder: BORROWER
-- Person who holds the item copy
reserver: BORROWER
-- Person who reserves the item
borrower: BORROWER
-- Person who borrows the item

reserved_end: DATE
-- Ending of reservation

Invariant

copy_status ∈ {"on shelf", "borrowed", "held", "reserved"};
new_status ∈ {"on shelf", "borrowed", "held", "reserved"};
∀ *p: ITEM | p.item_status=*reserved → *new_status=*held **and**
new_status ≠ borrowed

BOOK CIRCULATION SYSTEM : A BON MODEL

FORMAL CLASS DESCRIPTION

BORROW_SCREEN +

Inherits: *SCREEN*

input_id
→ *borrower_id: VALUE*
input_catalog
→ *catalog_no:VALUE*
input_date
→ *date_borrow:DATE*

OVERDUE_SCREEN +

Inherits: *SCREEN*

input_id
→ *borrower_id: VALUE*
input_payment
→ *amount_paid:VALUE*
input_date
→ *date_paid:DATE*

RETURN_SCREEN +

Inherits: *SCREEN*

input_catalog
→ *catalog_no:VALUE*
input_date
→ *date_return:DATE*

SEARCH_SCREEN +

Inherits: *SCREEN*

input_search_key
→ *search_key: SET[VALUE]*

QUERY_SCREEN +

Inherits: *SCREEN*

SCREEN *

is_open: BOOLEAN
-- Is menu open?

open
forth
back
select
accept
cancel
delete

Input_name
→ *user: BORROWER*

Input_password
→ *password: STRING*

BOOK CIRCULATION SYSTEM : A BON MODEL

FORMAL CLASS DESCRIPTION

BOOK_CIRCULATION

library, school: VALUE
menu: LIST[SCREEN]
collection:LIST[ITEM]
member:LIST[BORROWER]
librarian:LIST[STAFF]

is_open: BOOLEAN
-- Is system open

input_password
→ *password : VALUE*

check_password: BOOLEAN
-- Validate password
! *Result → password = sys_password*

activate system
close down system

TRANSACTION •

transit: VALUE
today, date_borrow, due_date, date_return, date_paid: DATE

person: BORROWER
-- Person who borrows item
librarian_b: STAFF
-- Librarian who facilitates borrowing transaction
librarian_r: STAFF
-- Librarian who facilitates returning transaction
librarian_o: STAFF
-- Librarian who facilitates overdue transaction
item_borrowed: ITEM_COPY
-- Item copy which is borrowed
is_overdue: BOOLEAN
-- Is the transaction overdue?
overdue_fees:NUMERIC
-- Overdue fees
compute_overdue:NUMERIC
? -- Compute overdue fees
! *is_overdue; person ≠ FACULTY;*
*overdue_fees=(today-due_date)*fine*
print_ovd_statement
-- Print overdue statement
print_transaction_info

Invariant

person ≠ Ø;
item_borrowed ≠ Ø;
due_date < date_borrow;
∀ p:person and (p : FACULTY or p : FACULTY) • due_date=end_semester;
∀ p:person and p : STUDENT•
due_date=date_borrow + 7;

APPENDIX L: QUESTIONNAIRE FOR SYSTEM ANALYSTS/DESIGNERS & PROGRAMMERS

Name : _____

Course : _____ **Year level** : _____

- Please choose one answer that is most suitable for you by crossing the letter in front of the answer. Fill in the blanks if asked to specify.
- Questions marked with **(*)** are applicable only for those who are familiar with structured programming.

QUESTIONS ON CONCEPTS

1.	Are you familiar with structured programming concepts such as function, procedure, etc.?	A. YES B. NO
2.	Are you familiar with object-oriented concepts such as class, object, encapsulation, inheritance, etc.?	A. YES B. NO
3.	What kind of structured programming languages are you familiar with ?	A. NONE B. C C. Pascal D. OTHERS:……
4.	What kind of object-oriented programming languages are you familiar with ?	A. C++ B. EIFFEL C. JAVA D. OTHERS: ……
5.	How long have you been using structured programming languages ? **(*)**	A. < 3 months B. 3 – 6 months C. 6 months – 1 year D. > 1 year
6.	How long have you been using object-oriented programming languages ?	A. < 3 months B. 3 – 6 months C. 6 months – 1 year D. > 1 year
7.	How do you asses your level of proficiency in object-oriented programming language ?	A. POOR B. AVERAGE C. GOOD D. EXCELLENT
8.	Are you familiar with the concepts of system analysis and design ?	A. YES B. NO
9.	How many times have you performed analysis & design for software development projects ?	A. 0 B. 1 – 5 times C. 6 – 10 times

		D. > 10 times
10.	Do you think object-oriented analysis and design should be different from structured analysis and design?	A. YES B. NO
11.	For those who learned structured programming before learning object-oriented programming, did you find any difficulties in switching from structured programming into object-oriented programming ? **(*)**	A. YES B. NO
12.	How do you find object-oriented programming compared to structured programming ? **(*)**	A. LESS POWERFUL B. NO SIGNIFICANT DIFFERENCES C. MORE POWERFUL

The questionnaires on the next pages, that are preceded either by charts or diagrams, are related to **Book Circulation System,** except one (i.e. the one related to DEPLOYMENT DIAGRAM). The description of the case study is elaborated on page 3. In this case study, we are assuming that you are one of :

1. **SYSTEM ANALYST/DESIGNER**
2. **PROGRAMMER**

in the development team who is going to build the system.

As part of the efforts to build the system, the team has come up with a set of charts and diagrams. Some of them are described in this questionnaire. The main purpose of this survey is to find out whether these charts and diagrams can be **easily understood**, **useful,** and **effective** both for you as the development team and for domain experts/users.

THEREFORE, IN ANSWERING THE QUESTIONS, PLEASE :

1. Carefully **examine** each chart and diagram.
2. Decide whether or not the chart/diagram is **understandable** for you.
3. Decide whether or not the chart/diagram is **useful** for you as SYSTEM ANALYST/DESIGNER and as PROGRAMMER as well.
4. Decide whether or not the chart/diagram is **effective** as a tool for you to fulfill your jobs as SYSTEM ANALYST/DESIGNER and as PROGRAMMER as well.
5. Give your **free comments** (if any) regarding the chart/diagram you are evaluating.
6. In the case of similar diagrams (one belongs to UML, whereas the other belongs to BON), **compare** the two diagrams and decide **your preference** (which one is more understandable and more effective).

NOTE:

- Some questions might not be applicable for you either as system analyst/designer or programmer. In this case, you will find "N/A" (NOT APPLICABLE) mark in the corresponding cell.

BOOK CIRCULATION SYSTEM: *PROBLEM DESCRIPTION*

In this system, faculty members and students borrow and return books. Only students get fined for books returned past the due date. Seven use cases/scenario are supported, the first three of which have the librarian as actor.

Use Case 1: Facilitate Borrow

The librarian has been given the borrower's ID card and the actual book. She first enters the ID number into the system and then the system validates this number and displays whether or not the ID number is valid and whether or not the borrower may borrow books (she may not if she has overdue books or if she has reached her borrowing limit). If the borrower is allowed to borrow a book, the librarian then enters the book catalog number. If the book exists and is available for borrowing, the system displays that the book has been recorded as borrowed.

Use case 2: Facilitate Return

The librarian has been given the actual book. She enters the book catalog number. If the book is valid, the system displays that the book has been recorded as returned and prints overdue information, if applicable.

Use case 3: Facilitate Overdue Payment

The librarian enters the ID number of a student. The system then displays an overdue statement for this student, if applicable. The librarian then confirms with the system that the student has paid the corresponding amount and the system displays that the student's record has been updated to reflect the paid fine.

Use case 4: Search Item

The borrower enters search information (title, subject, or author). The system then list all books that apply. The system then prints out all pertinent information (catalog number, whether or not the book is available, etc.)

Use case 5: Hold Item

A borrower may cause a book to be held for a full day, signifying her intention to borrow the book. This is done by simply pressing some key after she has located the book using scenario to search book described above. The borrower will be prompted for a name and a password before performing this transaction.

Use case 6: Reserve Item

Using a process similar to the process of intention to borrow books, a faculty member may place a book on reserve for a specified duration. Books on reserve may not be borrowed. A name and a password are also required before performing this transaction.

Use case 7: Query on Transaction

After giving a name and a password, a borrower may find out from the system which books she has borrowed and when these are due.

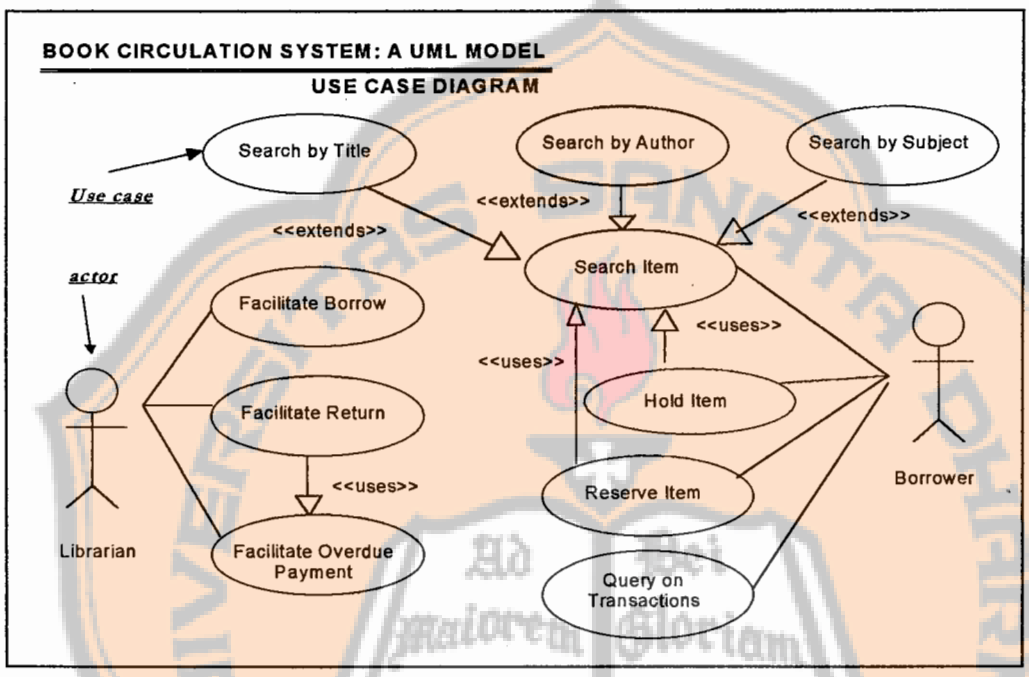
USE CASE DIAGRAM

Purpose of the diagram is to exhibit general description of the system in terms of actors, use cases, and the relationships between them.

Elements of a use case diagram:

- *Actor* : a role that a user plays with respect to the system
- *Use case* : a typical interaction between a user and a computer system (sometimes also called as scenario)
- *Relationship between actor & use case*
- *Relationships between use cases:*
  1. *Extends* : one use case similar to another use case but does a bit more
  2. *Uses* : one use case uses another use case whose one chunk of its behavior is similar to the behavior of the other use case.

Example of use case diagram:



Evaluations:

QUESTION	SYSTEM ANALYST/ DESIGNER	PROGRAMMER
Understandable ?	A. YES B. NO	A. YES B. NO
Useful ?	A. YES B. NO	A. YES B. NO
Effective to exhibit general description of the system ?	A. YES B. NO	A. YES B. NO
Comments		

CLUSTER CHART

Purpose of the chart:

- To briefly specify description of each class and subcluster in the cluster.

Example of cluster chart:

CLUSTER	*BOOK CIRCULATION SYSTEM*	**Part:** 1/1
PURPOSE System to facilitate borrowing and returning books, facilitate overdue payments, and keep track of book circulation in a library.		**INDEXING** **keywords:** book circulation system, selected and grouped classes

Class/(Clusters)	Description
BOOK_CIRCULATION	Root class containing system for managing book circulation in a library
BORROWER	Deferred class containing persons who are allowed to borrow item
ITEM	Deferred class containing items available to borrow
ITEM_COPY	Class containing copies of items available
SCREEN	Deferred class containing options and data entry
TRANSACTION	Record of every transaction made (borrow, return, overdue payment)
(BORROWERS)	Subcluster containing classes inheriting from class *BORROWER*
(ITEMS)	Subcluster containing classes inheriting from class ITEM
(SCREENS)	Subcluster containing classes inheriting from class SCREEN

Evaluations:

QUESTION	SYSTEM ANALYST/ DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to briefly specify description of classes & subclusters in the system ?	A. YES	B. NO	A. YES	B. NO
Effective to enhance communication with domain experts/users ?	A. YES	B. NO	N/A	
Do you think a **textual description** like CLUSTER CHART is more understandable than **graphical description** such as USE CASE DIAGRAM on the previous page ?	A. YES	B. NO	A. YES	B. NO
Comments				

CLASS CHART

The purpose of the chart is to specify definition of classes in terms of :

* *Commands* : what services can other classes ask the class to provide ?
* *Queries* : what information can other classes ask from the class ?
* *Constraints* : what rules must be obeyed by the class and its clients ?

Examples of class chart :

CLASS	*BORROWER*	**Part: 1/1**
TYPE OF OBJECT Person who is allowed to borrow items	**INDEXING** **Cluster:** *BOOK_CIRCULATION_SYSTEM* **Keywords:** book circulation system, borrower	
Queries	ID number. Name. Password. Total borrow. Borrower status (allowed/not allowed to borrow).	
Commands	Change status.	
Constraints	The ID number must be unique. Borrower is not allowed to borrow if she has overdue books or if she has reached borrowing limit.	

CLASS	*FACULTY*	Part: 1/1
TYPE OF OBJECT Faculty member who is allowed to borrow items	**INDEXING** **cluster:** *BORROWERS* **keywords:** faculty	
Inherits from	*BORROWER*	
Queries	Department	
Constraints	Borrowing limit = 25	

Evaluations:

QUESTION	SYSTEM ANALYST/ DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to specify definition of a class ?	A. YES	B. NO	A. YES	B. NO
Effective to help you coming up with robust design of classes ?	A. YES	B. NO	N/A	
Effective to enhance communication with domain experts/users ?	A. YES	B. NO	N/A	
Comments				

CLASS DIAGRAM, STATIC ARCHITECTURE & FORMAL CLASS DESCRIPTION

UML provides a CLASS DIAGRAM to describe the types of objects in the system and the various kind of relationships that exist among them. The same kind of diagram in BON is called as STATIC ARCHITECTURE.

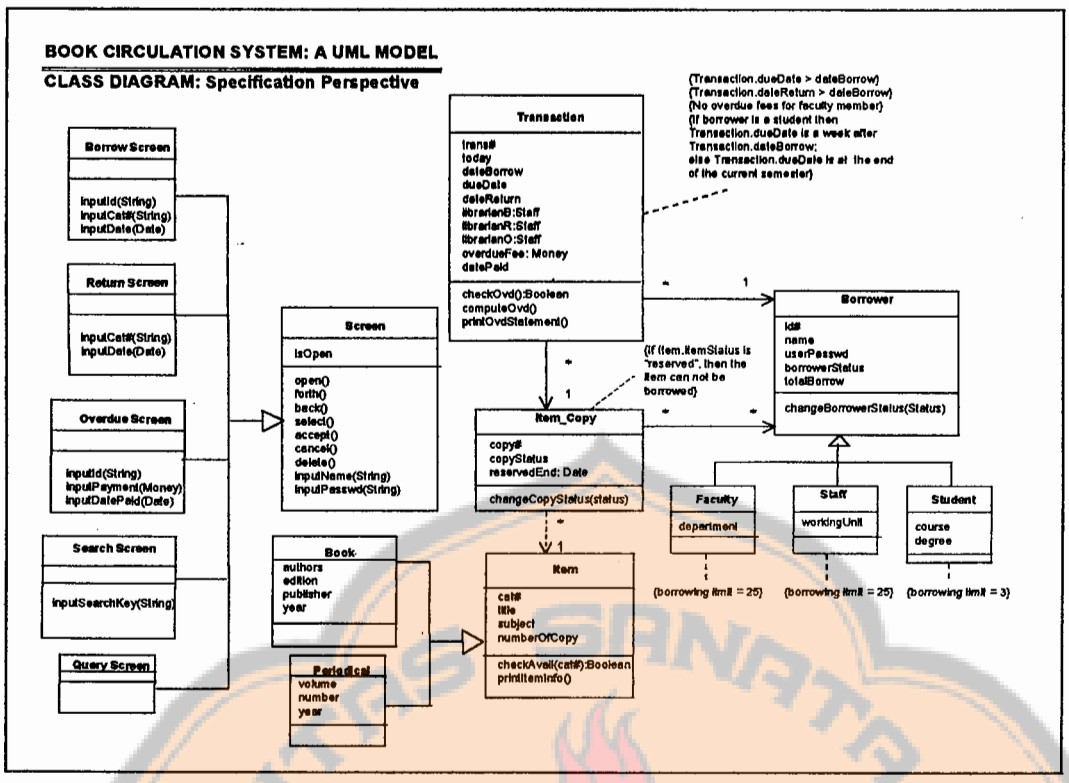
Elements of CLASS DIAGRAM:

- *Class* : a description of certain kinds of objects. It is described as a rectangle divided into 3 sections: class name, attributes, & operations
- *Relationship between classes*:
  1. Association : a relationship denoting a logical dependency between two classes
  2. Inheritance : a relationship in which one class shares the attribute and/or operation of other class(es).

3. Dependency: a relationship in which changes to the definition of one class may cause changes to the other.

- *Constraint* : particular condition that should be fulfilled by a class.
- *Multiplicity* : an indication of how many objects may participate in the given relationship.

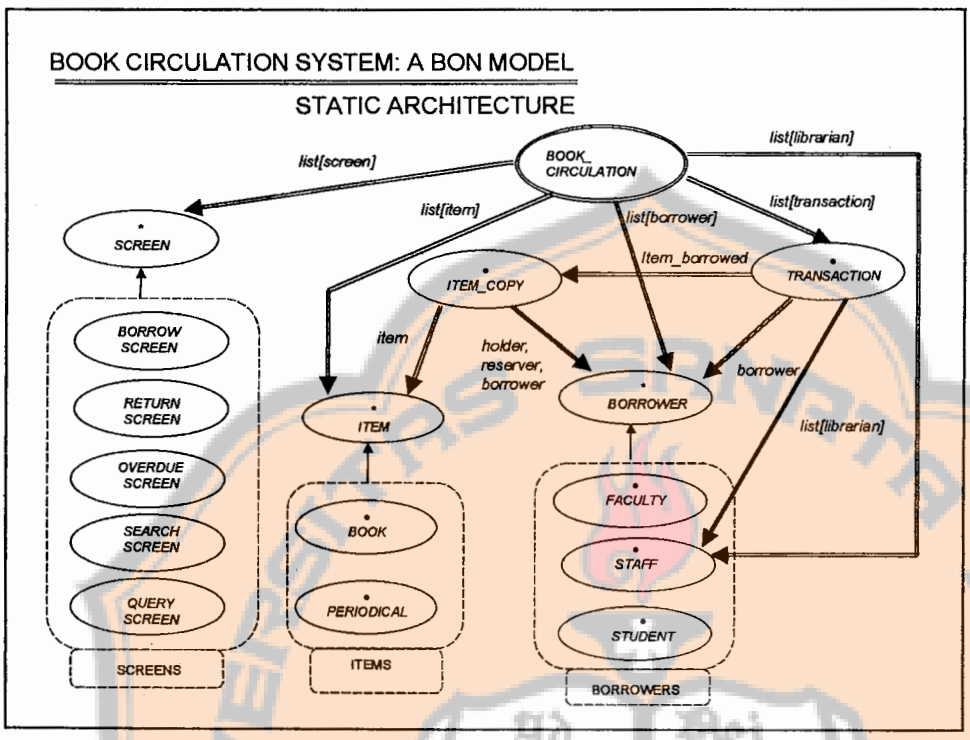
Example of class diagram:



Elements of STATIC ARCHITECTURE:

- *Class* : a description of a certain kinds of objects. It is described as an ellipse containing the class name and its type (e.g. abstract/deferred class, effective class, persistent class, etc.)
- *Cluster* : a group of classes
- *Relationship between classes/clusters*:
  1. *Client relation* : a relationship in which one class (client) uses services supplied by another class (supplier)
  2. *Inheritance* : a relationship in which one class (child) shares features (i.e. attributes and operations) of other class/classes (parent).
- *Role* : the purpose for which a certain class is used.

Example of static architecture:

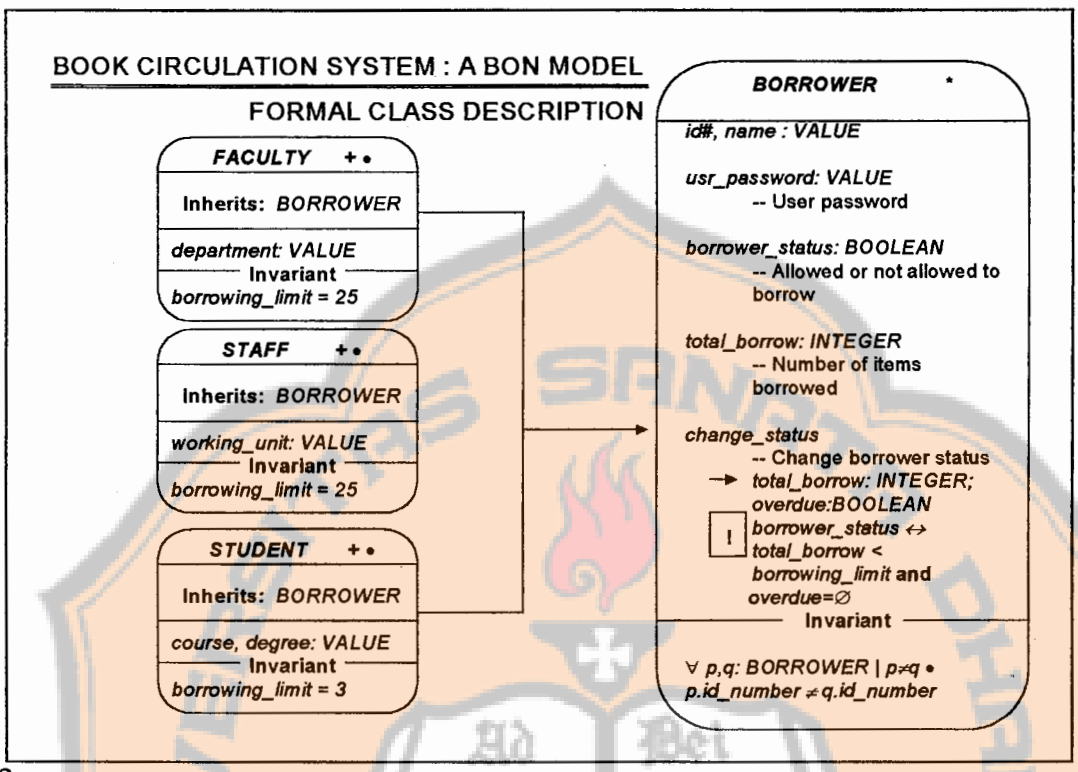


In an overview level of STATIC ARCHITECTURE, a class is described as an ellipse. In a more detail level, a more formal description of classes is used. This FORMAL CLASS DESCRIPTION consists of:

- *Class header* : class name with an annotation
- *Indexing clause* : general information about a class to be used for browsing & management
- *Inheritance clause* : parent classes from which the class inherits its features
- *Class features* : description of variables & operations of the class that may contain an optional *pre-condition* (i.e. a condition that should be fulfilled before an operation is executed) and an optional *post-condition* (i.e. a condition that should be fulfilled after an operation is executed)
- *Class invariant* : conditions that should be fulfilled in any state of the object.

Examples of formal class description:

1.



2.

2.

BOOK CIRCULATION SYSTEM : A BON MODEL

FORMAL CLASS DESCRIPTION

ITEM_COPY •

cat#: VALUE
copy_number:INTEGER

copy_status:VALUE
-- On shelf/borrowed/held/reserved

change_copy_status
-- Change item copy status
? *new_status: STATUS*
new_status="reserved" • *reserver:FACULTY;*
copy_status="held" **and** *new_status="borrowed"* •
borrower=holder
! *copy_status=new_status*

holder: BORROWER
-- Person who holds the item copy
reserver: BORROWER
-- Person who reserves the item
borrower: BORROWER
-- Person who borrows the item

reserved_end: DATE
-- Ending of reservation

Invariant

copy_status ∈ {"on shelf", "borrowed", "held", "reserved"};
new_status ∈ {"on shelf", "borrowed", "held", "reserved"};
∀ *p: ITEM* | *p.item_status*=reserved → *new_status*≠held **and** *new_status* ≠ *borrowed*

Evaluations:

A. CLASS DIAGRAM

QUESTION	SYSTEM ANALYST/ DESIGNER	PROGRAMMER
Understandable ?	A. YES B. NO	A. YES B. NO
Useful ?	A. YES B. NO	A. YES B. NO
Effective to describe the static structure of the system ?	A. YES B. NO	A. YES B. NO
Comments		

B. STATIC ARCHITECTURE

QUESTION	SYSTEM ANALYST/ DESIGNER	PROGRAMMER
Understandable ?	A. YES B. NO	A. YES B. NO
Useful ?	A. YES B. NO	A. YES B. NO
Effective to describe the **general static structure** of the system ?	A. YES B. NO	A. YES B. NO
Comments		

C. FORMAL CLASS DESCRIPTION

QUESTION	SYSTEM ANALYST/ DESIGNER	PROGRAMMER
Understandable ?	A. YES B. NO	A. YES B. NO
Useful ?	A. YES B. NO	A. YES B. NO
Effective to describe the **detail static structure** of the system?	A. YES B. NO	A. YES B. NO
Comments		

D. COMPARISON

QUESTION	SYSTEM ANALYST/ DESIGNER	PROGRAMMER
Which diagram is more understandable ?	A. CLASS DIAGRAM B. STATIC ARCHITECTURE + FORMAL CLASS DESCR. C. NO PREFERENCE (SAME) D. NONE OF THEM because	A. CLASS DIAGRAM B. STATIC ARCHITECTURE + FORMAL CLASS DESCR. C. NO PREFERENCE (SAME) D. NONE OF THEM because .....
Which diagram is more effective to describe the static architecture of the system ?	A. CLASS DIAGRAM B. STATIC ARCHITECTURE + FORMAL CLASS DESCR. C. NO PREFERENCE (SAME) D. NONE OF THEM because	A. CLASS DIAGRAM B. STATIC ARCHITECTURE + FORMAL CLASS DESCR. C. NO PREFERENCE (SAME) D. NONE OF THEM because .....
Which diagram is more effective to help you during **implementation stage** ?	N/A	A. CLASS DIAGRAM B. STATIC ARCHITECTURE + FORMAL CLASS DESCR. C. NO PREFERENCE (SAME) D. NONE OF THEM because .....
Comments		

OBJECT SCENARIO & COLLABORATION DIAGRAM

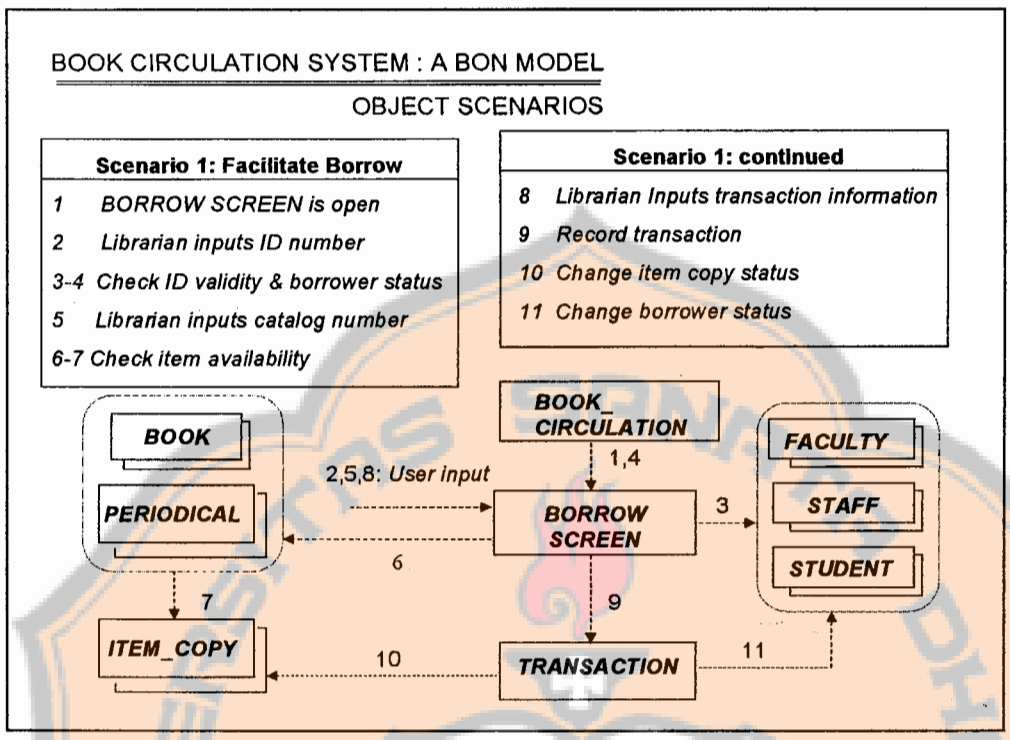
Purpose of the diagram:

- To describe how group of objects collaborate (in a single use case/scenario)

Elements of OBJECT SCENARIO:

- *Scenario box* : description of a possible system execution
- *Object/multiple objects* : basic component of the system
- *Message link* : represents message sent from one object to another
- *The sequence number of message* which correspond to entries in the scenario box.

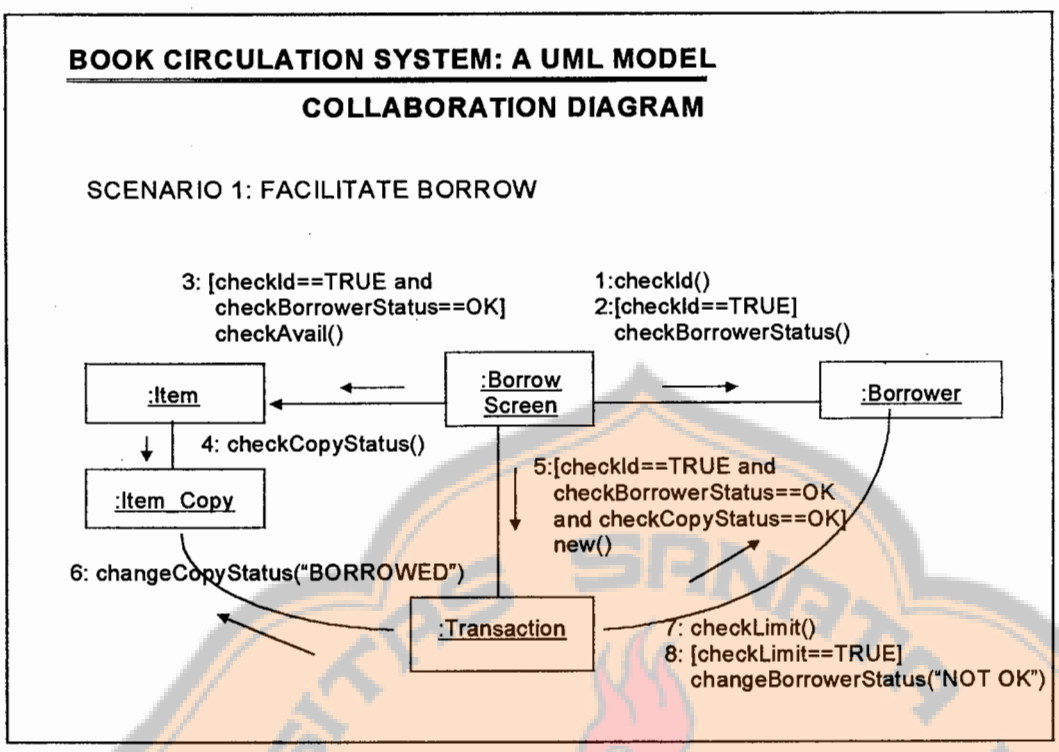
Example of object scenario:



Elements of COLLABORATION DIAGRAM:

- *Object* : basic component of the system
- *Message* : represents message sent from one object to another. A message may have a **condition** written inside square brackets which indicate that the message is only sent if the condition is TRUE.
- *The sequence number of message*

Example of collaboration diagram:



Evaluations:

A. COLLABORATION DIAGRAM

QUESTION	SYSTEM ANALYST/ DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to describe object collaboration ?	A. YES	B. NO	A. YES	B. NO
Comments				

B. OBJECT SCENARIO

QUESTION	SYSTEM ANALYST/ DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to describe object collaboration ?	A. YES	B. NO	A. YES	B. NO
Comments				

C. COMPARISON

QUESTION	SYSTEM ANALYST/ DESIGNER	PROGRAMMER
Which diagram is more understandable ?	A. COLLABORATION DIAGR. B. OBJECT SCENARIO C. NO PREFERENCE (SAME) D. NONE OF THEM because	A. COLLABORATION DIAGR. B. OBJECT SCENARIO C. NO PREFERENCE (SAME) D. NONE OF THEM because .....
Which diagram is more effective to describe object collaboration ?	A. COLLABORATION DIAGR. B. OBJECT SCENARIO C. NO PREFERENCE (SAME) D. NONE OF THEM because	A. COLLABORATION DIAGR. B. OBJECT SCENARIO C. NO PREFERENCE (SAME) D. NONE OF THEM because .....
Which diagram is more effective to help you during implementation ?	N/A	A. COLLABORATION DIAGR. B. OBJECT SCENARIO C. NO PREFERENCE (SAME) D. NONE OF THEM because .....
Comments		

ACTIVITY DIAGRAM

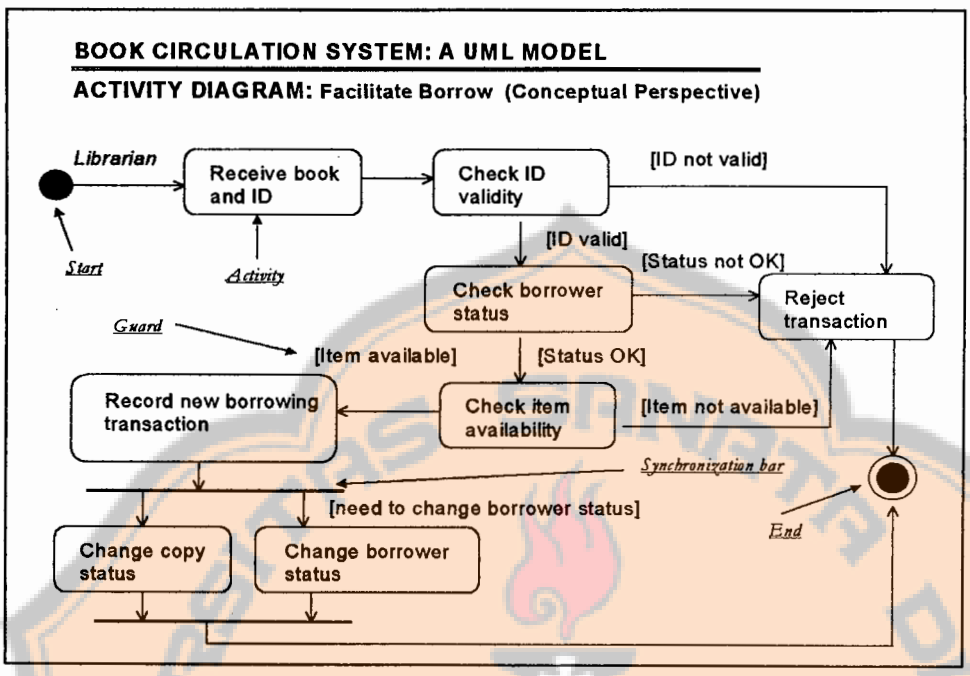
Purpose of the diagram:

- To show object behavior within control structure.

Elements of ACTIVITY DIAGRAMS:

- *Activity* : some task that needs to be done or a method on a class.
- *Start point*
- *End point*
- *Guard* : a logical expression that evaluates to "true" or "false", attached to an arrow showing the flow of process.
- *Synchronization bar* : a bar to which several outgoing triggers are attached to, showing that the activities can occur in parallel.

Example of activity diagram:



Evaluations:

QUESTION	SYSTEM ANALYST/ DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to describe object behavior within control structure ?	A. YES	B. NO	A. YES	B. NO
Comments				

STATE DIAGRAM

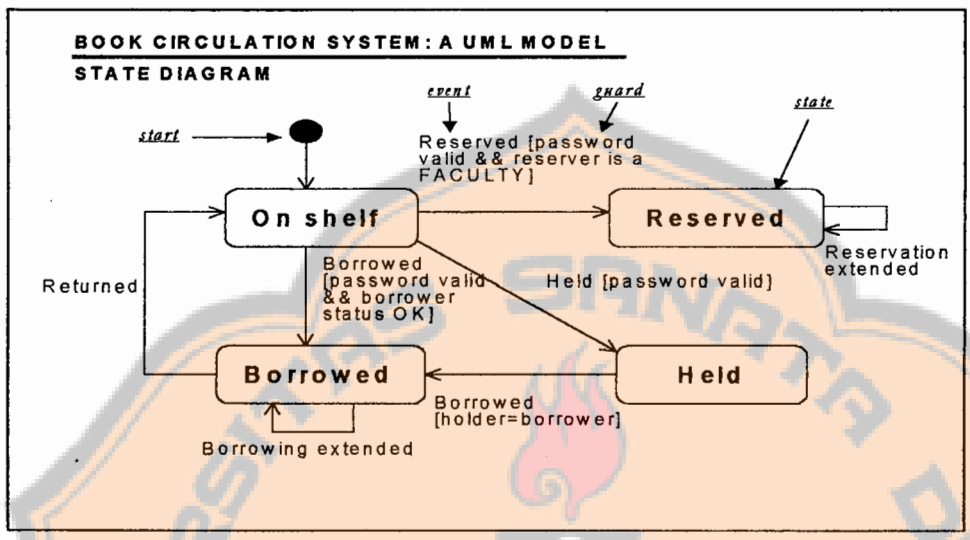
Purpose of the diagram:

- To show object behavior across many use cases.

Elements of a state diagram:

- *State* : a possible circumstance that a particular object can get into
- *Transition* : a transformation from one state to another. A transition has a label which consists of three optional parts: *Event[Guard]/Action*
  - * *Event* : a circumstance that triggers the transition to proceed.
  - * *Guard* : a logical condition that will return TRUE or FALSE. A guarded transition occurs only if the guard resolves to TRUE.
  - * *Action* : a task associated with transitions
- *Activity* : a task associated with a state.
- *Start state*

Example of state diagram:



Evaluations:

QUESTION	SYSTEM ANALYST/ DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to describe object behavior across many use cases ?	A. YES	B. NO	A. YES	B. NO
Comments				

PACKAGE DIAGRAM

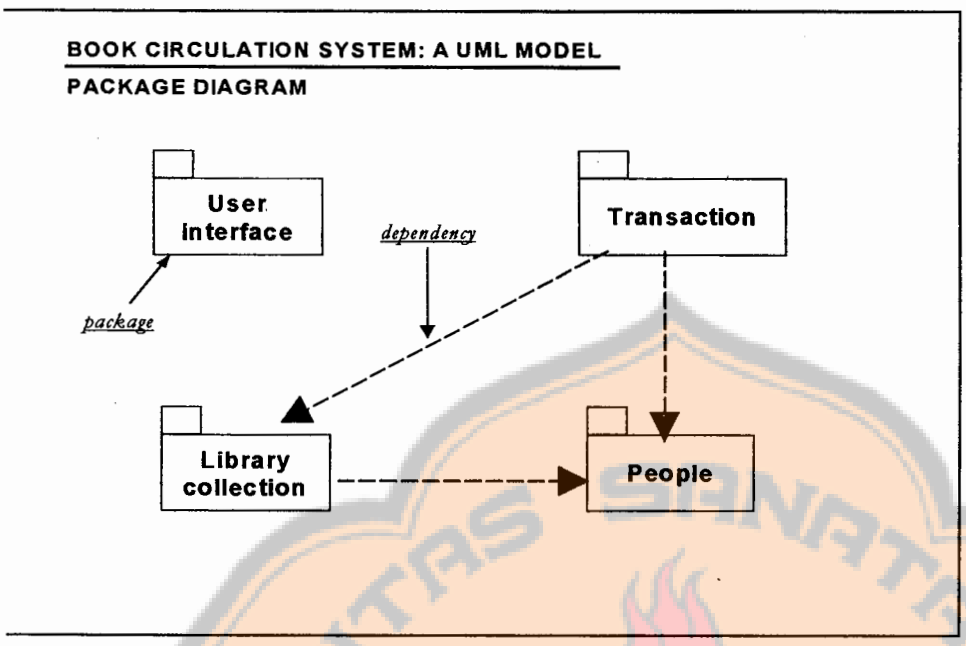
Purpose of the diagram:

- To show packages of classes and dependencies among them.

Elements of a package diagram:

- *Package* : group of classes
- *Dependency* : a relationship between two elements in which changes to the definition of one element may cause changes to the other.

Example of package diagram:



Evaluations:

QUESTION	SYSTEM ANALYST/ DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to describe packages of classes and dependencies among them ?	A. YES	B. NO	A. YES	B. NO
Comments				

DEPLOYMENT DIAGRAM

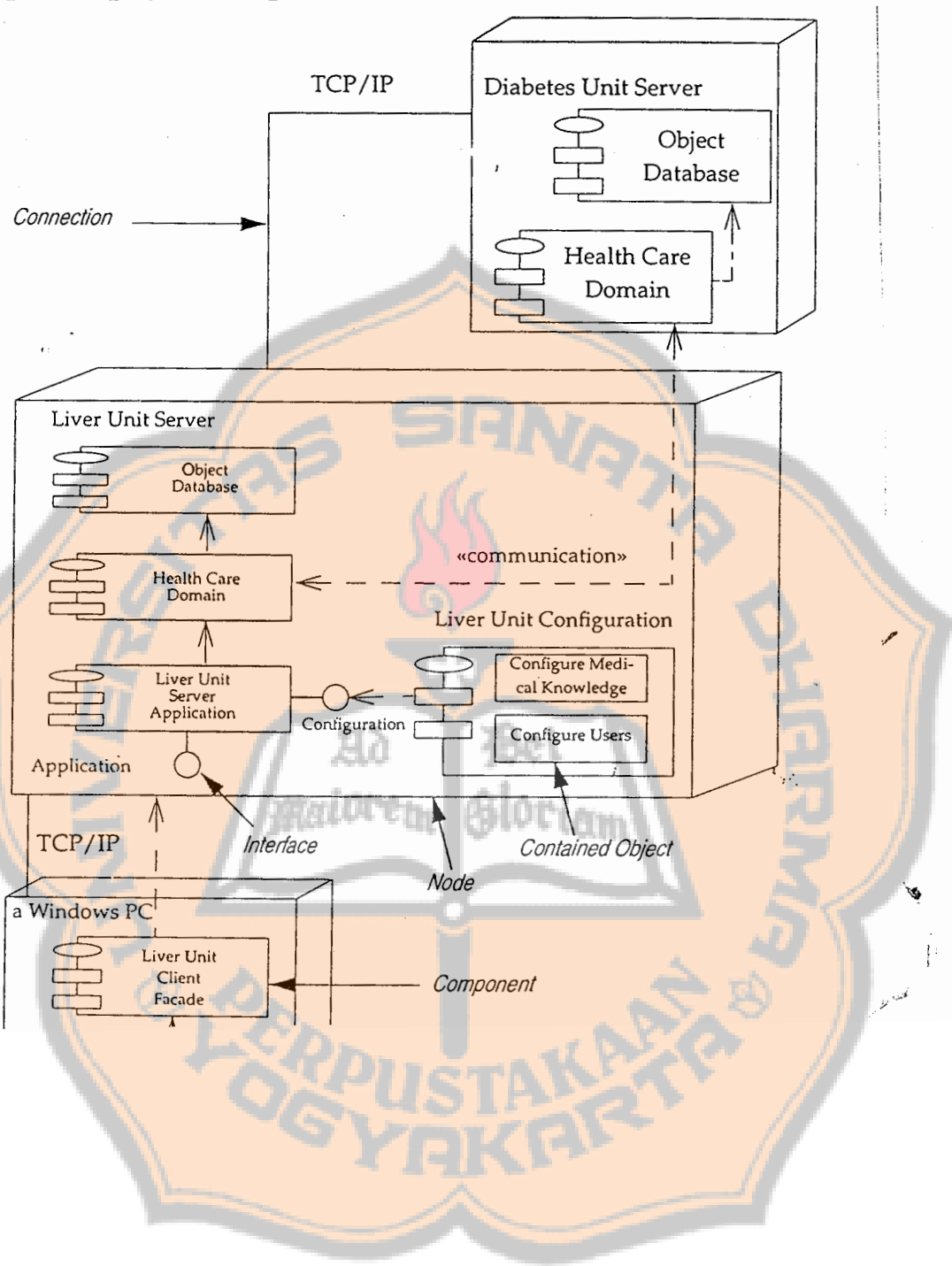
Purpose of the diagram:

- To show deployment of components on hardware nodes.

Elements of a deployment diagram:

- *Node* : some kind of computational unit
- *Component* : physical modules of code
- *Connection* : the communication path over which the nodes of the system will interact
- *Interface* : the communication path between components

Example of deployment diagram:



Evaluations:

QUESTION	SYSTEM ANALYST / DESIGNER		PROGRAMMER	
Understandable ?	A. YES	B. NO	A. YES	B. NO
Useful ?	A. YES	B. NO	A. YES	B. NO
Effective to describe deployment of software components on hardware nodes ?	A. YES	B. NO	A. YES	B. NO
Comments				

APPENDIX M: QUESTIONNAIRE FOR DOMAIN EXPERTS/USERS

Name : _____

Position : _____

Please choose one answer that is most suitable for you by crossing the letter in front of the answer. Fill in the blanks if asked to specify.

1.	How long have you been working as a staff in the library ?	.....
2.	How do you asses your level of proficiency in computer literacy ?	A. POOR B. AVERAGE C. GOOD D. EXCELLENT
3.	Have you ever been involved in any effort of computerization, such as developing of computerized library circulation system ?	A. YES B. NO
4.	Are you familiar with any kind of programming languages ?	A. YES (please specify the programming languages) ..... B. NO

The questionnaires on the next pages that are preceded either by charts or diagrams are related to **Book Circulation System**. In this case study, we are assuming that the library where you are working hires a team to computerize book circulation system, which is still done manually.

As part of their efforts to understand the library system and to fulfill your expectations, the team has drawn a set of charts and diagrams. In order to verify whether their charts and diagrams reflect what you want, the team gives the charts and diagrams to you. Thus, these charts and diagrams will likely serve as a language between you and the team during the discussion about the book circulation system.

The main purpose of this survey is to find out whether the charts and diagrams can be **easily understood** and **useful** both for you as users and for the team. Therefore, in answering the questions, please :

1. Carefully **examine** each chart and diagram.
2. Decide whether or not the charts/diagrams is **understandable** for you.
3. Decide whether or not they are **useful** for you to identify and describe what you want.
4. In the case of similar diagrams, **compare** the two diagrams and decide your preference.

BOOK CIRCULATION SYSTEM: *PROBLEM DESCRIPTION*

In this system, faculty members and students borrow and return books. Only students get fined for books returned past the due date. Seven use cases/scenario are supported, the first three of which have the librarian as actor.

Use Case 1: Facilitate Borrow

The librarian has been given the borrower's ID card and the actual book. She first enters the ID number into the system and then the system validates this number and displays whether or not the ID number is valid and whether or not the borrower may borrow books (she may not if she has overdue books or if she has reached her borrowing limit). If the borrower is allowed to borrow a book, the librarian then enters the book catalog number. If the book exists and is available for borrowing, the system displays that the book has been recorded as borrowed.

Use case 2: Facilitate Return

The librarian has been given the actual book. She enters the book catalog number. If the book is valid, the system displays that the book has been recorded as returned and prints overdue information, if applicable.

Use case 3: Facilitate Overdue Payment

The librarian enters the ID number of a student. The system then displays an overdue statement for this student, if applicable. The librarian then confirms with the system that the student has paid the corresponding amount and the system displays that the student's record has been updated to reflect the paid fine.

Use case 4: Search Item

The borrower enters search information (title, subject, or author). The system then lists all books that apply. The system then prints out all pertinent information (catalog number, whether or not the book is available, etc.)

Use case 5: Hold Item

A borrower may cause a book to be held for a full day, signifying her intention to borrow the book. This is done by simply pressing some key after she has located the book using scenario to search book described above. The borrower will be prompted for a name and a password before performing this transaction.

Use case 6: Reserve Item

Using a process similar to the process of intention to borrow books, a faculty member may place a book on reserve for a specified duration. Books on reserve may not be borrowed. A name and a password are also required before performing this transaction.

Use case 7: Query on Transaction

After giving a name and a password, a borrower may find out from the system which books she has borrowed and when these are due.

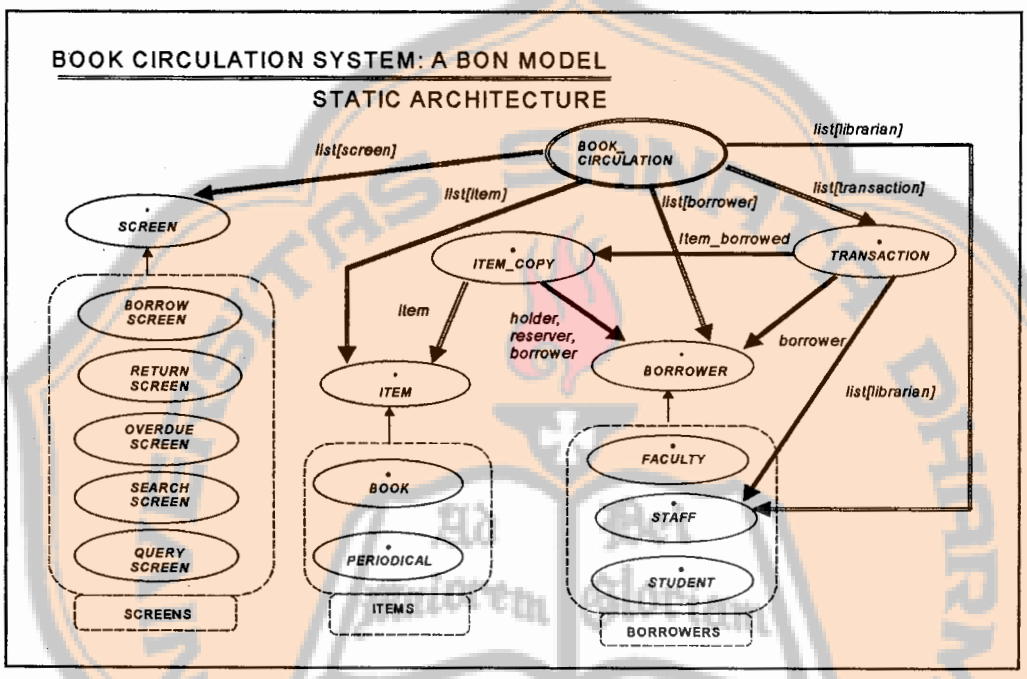
USE CASE DIAGRAM

Use case diagram is a graphical representation of actors, use cases, and the relationships between them.

Elements of a use case diagram:

- *Actor* : a role that a user plays with respect to the system
- *Use case* : a typical interaction between a user and a computer system
- *Relationship between actor & use case*
- *Relationships between use cases:*
  1. *Extends* : one use case similar to another use case but does a bit more
  2. *Uses* : one use case uses another use case whose one chunk of its behavior is similar to the behavior of the other use case.

Example of use case diagram:



Questions:

1.	How do you find the USE CASE DIAGRAM ?	A. EASY TO UNDERSTAND B. DIFFICULT TO UNDERSTAND because ....
2	Will CLASS DIAGRAM be useful for you to verify the team's understanding of the library system ?	A. YES B. NO

CLUSTER CHART

Purpose of the chart:

- To briefly specify description of each class and subcluster in the cluster.

Class is an object in the library system.
Cluster is a group of classes.

Example of cluster chart:

CLUSTER	*BOOK CIRCULATION SYSTEM*	**Part:** 1/1
PURPOSE System to facilitate borrowing and returning books, facilitate overdue payments, and keep track of book circulation in a library.		**INDEXING** **keywords:** book circulation system, selected and grouped classes

Class/(Clusters)	Description
BOOK_CIRCULATION	Root class containing system for managing book circulation in a library
BORROWER	Deferred class containing persons who are allowed to borrow item
ITEM	Deferred class containing items available to borrow
ITEM_COPY	Class containing copies of items available
SCREEN	Deferred class containing options and data entry
TRANSACTION	Record of every transaction made (borrow, return, overdue payment)
(BORROWERS)	Subcluster containing classes inherited from class *BORROWER*
(ITEMS)	Subcluster containing classes inherited from class ITEM
(SCREENS)	Subcluster containing classes inherited from class SCREEN

Questions:

1.	How do you find the CLUSTER CHART ?	A. EASY TO UNDERSTAND B. DIFFICULT TO UNDERSTAND because (..... .....
2	Supposedly you are a user who is getting involved in a library computerization effort, will CLUSTER CHART be useful for you to identify and describe all significant components of the library system ?	A. YES B. NO
3.	Do you think this kind of chart (textual form) is more understandable for you compared to a description in graphical form (like USE CASE DIAGRAM on the previous page) ?	A. YES B. NO

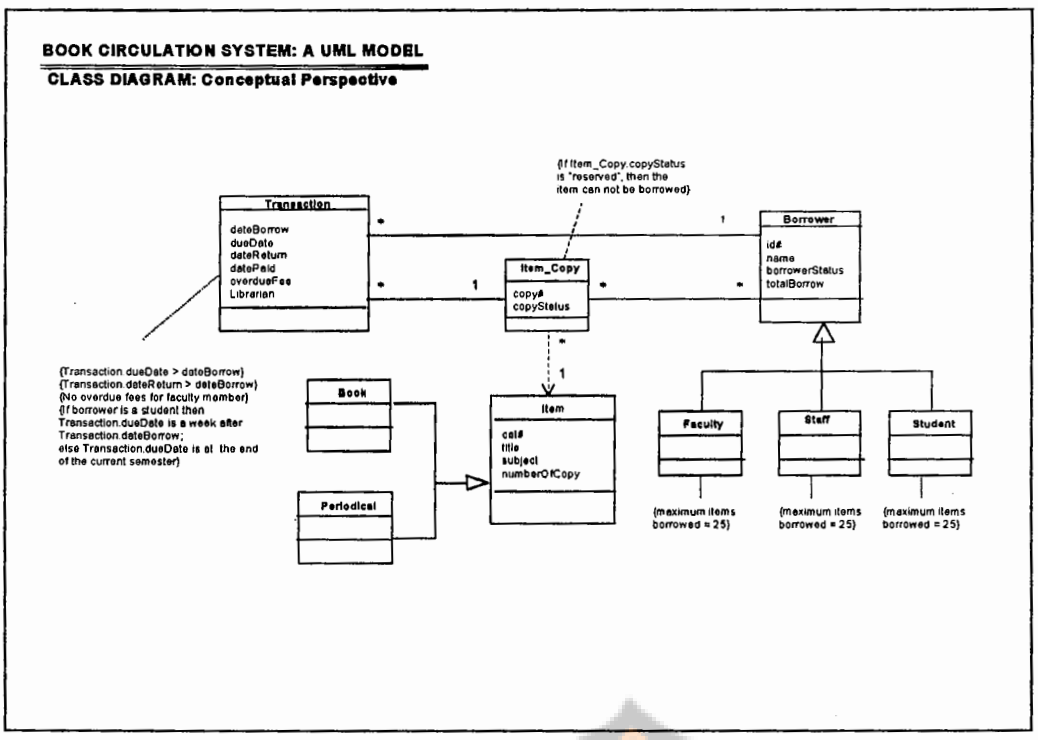
CLASS DIAGRAM, STATIC ARCHITECTURE & FORMAL CLASS DESCRIPTION

UML provides a CLASS DIAGRAM to describe the types of objects in the system and the various kinds of relationships that exist among them. The same kind of diagram in BON is called STATIC ARCHITECTURE.

Elements of CLASS DIAGRAM:

- *Class* : a description of certain kinds of objects. It is described as a rectangle divided into 3 sections: class name, attributes, & operations
- *Relationship between classes*:
  1. Association : a relationship denoting a logical dependency between two classes
  2. Inheritance : a relationship in which one class shares the attribute and/or operation of other class(es).
  3. Dependency: a relationship in which changes to the definition of one class may cause changes to the other.
- *Constraint* : particular condition that should be fulfilled by a class.
- *Multiplicity* : an indication of how many objects may participate in the given relationship.

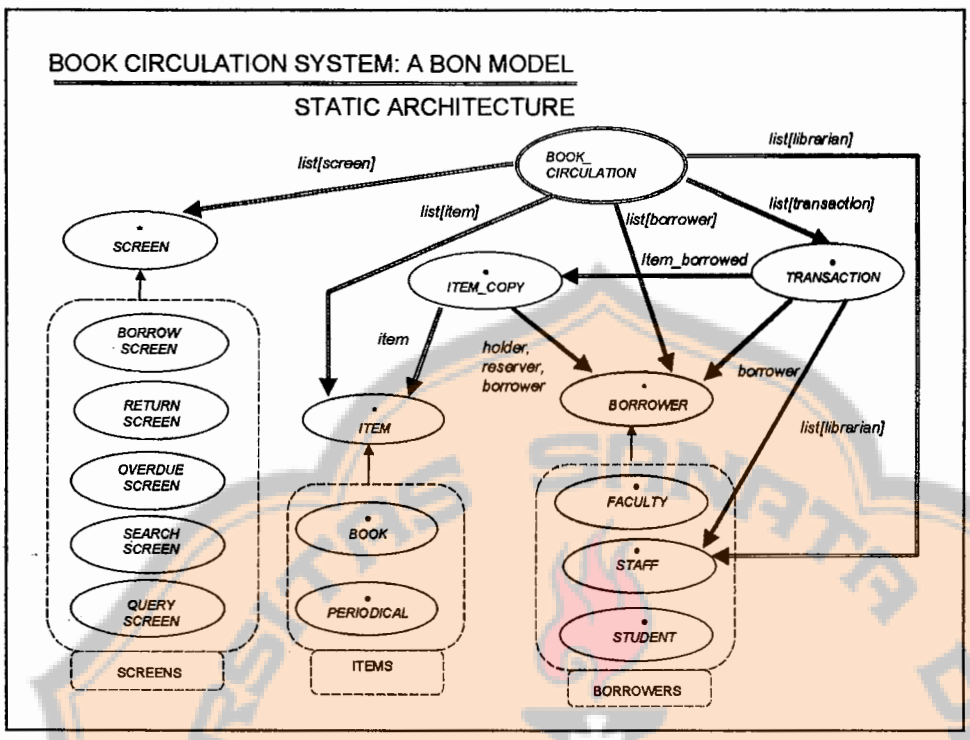
Example of class diagram:



Elements of STATIC ARCHITECTURE:

- *Class* : a description of a certain kinds of objects. It is described as an ellipse containing the class name and its type (e.g. abstract/deferred class, effective class, persistent class, etc.)
- *Cluster* : a group of classes
- *Relationship between classes/clusters*:
  1. *Client relation* : a relationship in which one class (client) uses services supplied by another class (supplier)
  2. *Inheritance* : a relationship in which one class (child) shares features (i.e. attributes and operations) of other class/classes (parent).
- *Role* : the purpose for which a certain class is used.

Example of static architecture:



Questions:

1.	How do you find the CLASS DIAGRAM ?	A. EASY TO UNDERSTAND B. DIFFICULT TO UNDERSTAND because ....
2.	How do you find the STATIC ARCHITECTURE?	A. EASY TO UNDERSTAND B. DIFFICULT TO UNDERSTAND because ....
3	Will CLASS DIAGRAM & STATIC ARCHITECTURE be useful for you to verify what the team understands about the library system & your expectations ?	A. YES B. NO
4	Which diagram do you prefer most to describe the structure of the library system ?	A. CLASS DIAGRAM B. STATIC ARCHITECTURE C. NO PREFERENCE (BOTH ARE THE SAME) D. NONE OF THEM because ....

ACTIVITY DIAGRAM

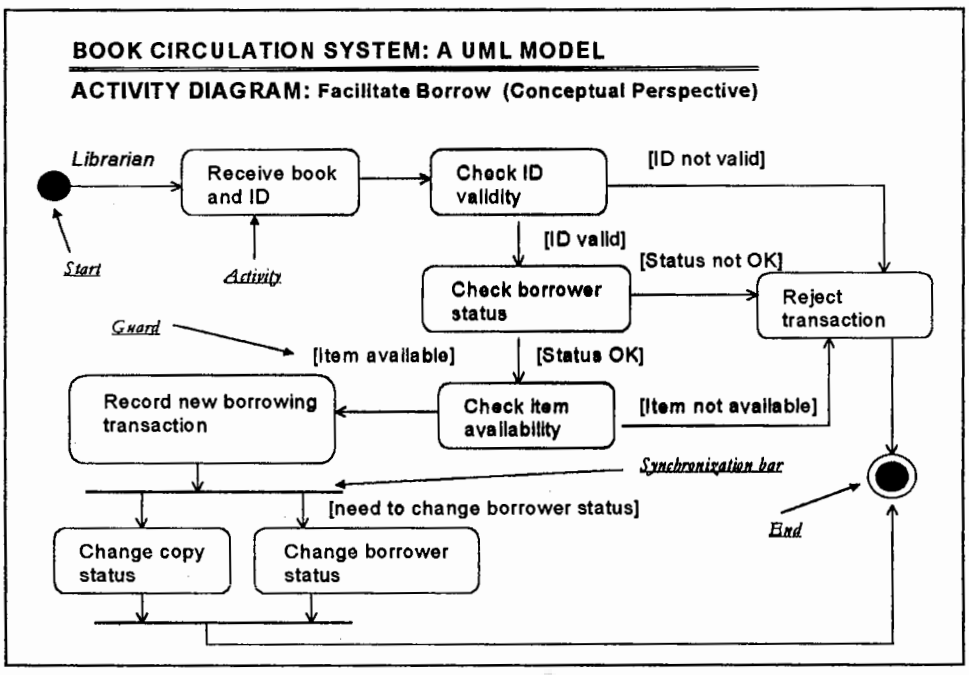
Purpose of the diagram:

- To show the process flow.

Elements of ACTIVITY DIAGRAMS:

- *Activity* : some task that needs to be done or a method on a class.
- *Start point*
- *End point*
- *Guard* : a logical expression that evaluates to "true" or "false", attached to an arrow showing the flow of process.
- *Synchronization bar* : a bar to which several outgoing triggers are attached to, showing that the activities can occur in parallel.

Example of activity diagram:



Questions:

1.	How do you find the ACTIVITY DIAGRAM ?	A. EASY TO UNDERSTAND B. DIFFICULT TO UNDERSTAND because ....
2	Will ACTIVITY DIAGRAM be useful for you to verify the team's understanding of the process flow in the library system ?	A. YES B. NO

APPENDIX N:

RESULTS OF THE SURVEY AMONG SYSTEM ANALYSTS/ DESIGNERS

Respondents Profile

Number of respondents = 10

1.	Course & year level a. MIS (3rd year) b. MS-CS	 7 3
2.	Familiar with structured programming a. Structured programming known b. Experience with structured programming	5 C, Pascal, Perl < 3 mos - > 1 year
3.	Familiar with OO concepts	10
4.	OOPL known a. C++ b. Java	 3 10
5.	Experience with OOPL a. < 3 months b. 3-6 months c. 6-12 months d. > 1 year	 - 3 2 5
6.	Level of proficiency in OOPL a. Average b. Good c. No answer	 3 6 1
7.	Familiar with SAD	10
8.	Experience with SAD a. 0 times b. 1-5 times c. 6-10 times d. > 10 times	 1 8 1

Results of the Survey

* 7/10 means 7 out of 10 respondents

1. USE CASE DIAGRAM

QUESTION	YES	NO
Understandable ?	10/10	–
Useful ?	9/10	1/10
Effective to exhibit general description of the system ?	9/10	1/10

2. CLUSTER CHART

QUESTION	YES	NO
Understandable ?	8/10	2/10
Useful ?	10/10	–
Effective to briefly specify the descriptions of classes and subclusters?	10/10	–
Effective to enhance communication with non-technical people ?	8/10	2/10
Is textual description more understandable than graphical description ?	3/10	7/10

3. CLASS CHART

QUESTION	YES	NO
Understandable ?	8/10	2/10
Useful ?	8/10	2/10
Effective to specify the definition of classes ?	8/10	2/10
Effective to help coming up with a robust design of classes ?	8/10	2/10
Effective to enhance communication with non-technical people ?	8/10	2/10

4. CLASS DIAGRAM

QUESTION	YES	NO
Understandable ?	8/10	2/10
Useful ?	9/10	1/10
Effective to describe the static architecture of the system ?	9/10	1/10

5. STATIC ARCHITECTURE

QUESTION	YES	NO
Understandable ?	8/10	2/10
Useful ?	8/10	2/10
Effective to describe the general static architecture of the system ?	8/10	2/10

6. FORMAL CLASS DESCRIPTION

QUESTION	YES	NO	NO ANSWER
Understandable ?	8/10	2/10	–
Useful ?	7/10	3/10	–
Effective to describe the detail static architecture of the system ?	9/10	–	1/10

7. COMPARISON: CLASS DIAGRAM AND STATIC ARCHITECTURE

QUESTION	CLASS DIAGRAM	STATIC ARCHI + FORMAL CLASS DESC.	NO PREFERENCE	NONE OF THEM
More understandable ?	4/10	5/10	1/10	–
More effective to describe static architecture ?	4/10	5/10	–	–

8. COLLABORATION DIAGRAM

QUESTION	YES	NO
Understandable ?	7/10	3/10
Useful ?	7/10	3/10
Effective to describe object collaboration ?	8/10	2/10

9. OBJECT SCENARIO

QUESTION	YES	NO
Understandable ?	8/10	2/10
Useful ?	8/10	2/10
Effective to describe object collaboration ?	9/10	1/10

10. COMPARISON: COLLABORATION DIAGRAM AND OBJECT SCENARIO

QUESTION	COLLA-BORATION DIAGRAM	OBJECT SCENARIO	NO PREFERENCE	NONE OF THEM
More understandable ?	1/10	6/10	1/10	2/10
More effective to describe object collaboration ?	3/10	5/10	2/9	–

11. ACTIVITY DIAGRAM

QUESTION	YES	NO
Understandable ?	10/10	–
Useful ?	10/10	–
Effective to describe object behavior within control structure ?	10/10	–

12. STATE DIAGRAM

QUESTION	YES	NO
Understandable ?	8/10	2/10
Useful ?	7/10	3/10
Effective to describe object behavior within control structure ?	7/10	3/10

13. PACKAGE DIAGRAM

QUESTION	YES	NO
Understandable ?	8/10	2/10
Useful ?	5/10	5/10
Effective to describe package of classes & dependencies among them?	6/10	4/10

14. DEPLOYMENT DIAGRAM

QUESTION	YES	NO	NO ANSWER
Understandable ?	5/10	5/10	–
Useful ?	6/10	4/10	–
Effective to describe deployment of software components on hardware nodes ?	5/10	4/10	1/10

APPENDIX O: RESULTS OF THE SURVEY AMONG PROGRAMMERS

Respondents Profile

Number of respondents = 7

1.	Course & year level a. MIS (3rd year) b. MS-CS	 4 3
2.	Familiar with structured programming a. Structured programming known b. Experience with structured programming	2 C, Pascal, Perl 3 mos - > 1 year
3.	Familiar with OO concepts	7
4.	OOPL known a. C++ b. Java	 3 7
5.	Experience with OOPL a. < 3 months b. 3-6 months c. 6-12 months d. > 1 year	 – 2 1 4
6.	Level of proficiency in OOPL a. Average b. Good c. No answer	 1 5 1
7.	Familiar with SAD	7
8.	Experience with SAD a. 0 times b. 1-5 times c. 6-10 times	 - 6 1

Results of the Survey

* 6/7 means 6 out of 7 respondents

1. USE CASE DIAGRAM

QUESTION	YES	NO
Understandable ?	6/7	1/7
Useful ?	6/7	1/7
Effective to exhibit general description of the system ?	7/7	–

2. CLUSTER CHART

QUESTION	YES	NO	NO ANSWER
Understandable ?	6/7	1/7	–
Useful ?	7/7	–	–
Effective to briefly specify the descriptions of classes and subclusters?	5/7	2/7	–
Is textual description more understandable than graphical description ?	4/7	2/7	1/7

3. CLASS CHART

QUESTION	YES	NO
Understandable ?	6/7	1/7
Useful ?	6/7	1/7
Effective to specify the definition of classes ?	6/7	1/7

4. CLASS DIAGRAM

QUESTION	YES	NO
Understandable ?	6/7	1/7
Useful ?	7/7	–
Effective to describe the static architecture of the system ?	7/7	–

5. STATIC ARCHITECTURE

QUESTION	YES	NO
Understandable ?	7/7	–
Useful ?	6/7	1/7
Effective to describe the general static architecture of the system ?	6/7	1/7

6. FORMAL CLASS DESCRIPTION

QUESTION	YES	NO	NO ANSWER
Understandable ?	6/7	–	1/7
Useful ?	6/7	–	1/7
Effective to describe the detail static architecture of the system ?	6/7	–	1/7

7. COMPARISON: CLASS DIAGRAM AND STATIC ARCHITECTURE + FORMAL CLASS DESCRIPTION

QUESTION	CLASS DIAGRAM	STATIC ARCHI + FORMAL CLASS DESC.	NO PREFERENCE	NONE OF THEM
More understandable ?	5/7	2/7	–	–
More effective to describe static architecture ?	5/7	2/7	–	–
More effective to help during implementation?	5/7	2/7	–	–

8. COLLABORATION DIAGRAM

QUESTION	YES	NO
Understandable ?	7/7	–
Useful ?	7/7	–
Effective to describe object collaboration ?	7/7	–

9. OBJECT SCENARIO

QUESTION	YES	NO
Understandable ?	6/7	1/7
Useful ?	5/7	2/7
Effective to describe object collaboration ?	6/7	1/7

10. COMPARISON: COLLABORATION DIAGRAM AND OBJECT SCENARIO

QUESTION	COLLA-BORATION DIAGRAM	OBJECT SCENARIO	NO PREFERENCE	NONE OF THEM
More understandable ?	3/7	3/7	1/7	–
More effective to describe object collaboration ?	5/7	–	2/7	–
More effective to help during implementation?	6/7	–	1/7	–

11. ACTIVITY DIAGRAM

QUESTION	YES	NO
Understandable ?	6/7	1/7
Useful ?	6/7	1/7
Effective to describe object behavior within control structure ?	7/7	–

12. STATE DIAGRAM

QUESTION	YES	NO
Understandable ?	5/7	2/7
Useful ?	7/7	–
Effective to describe object behavior within control structure ?	6/7	–

13. PACKAGE DIAGRAM

QUESTION	YES	NO
Understandable ?	4/7	3/7
Useful ?	3/7	4/7
Effective to describe package of classes & dependencies among them?	3/7	4/7

14. DEPLOYMENT DIAGRAM

QUESTION	YES	NO
Understandable ?	5/7	2/7
Useful ?	7/7	–
Effective to describe deployment of software components on hardware nodes ?	4/7	2/7

APPENDIX P:

RESULTS OF THE SURVEY AMONG DOMAIN EXPERTS/USERS

Respondents Profile

Number of respondents = 5

1.	Position:	
	a. Chief of circulation section	1
	b. Library assistant	2
	c. Technical assistant	2
2.	Working experience:	
	a. 1-5 years	1
	b. 6-10 years	1
	c. 11-15 years	1
	d. > 15 years	2
3.	Level of proficiency in computer literacy	
	a. Average	3
	b. Good	2
4.	Posses experiences in computerization process?	
	a. Yes	4
	b. No	1
5.	Familiar with programming languages?	
	a. Yes	2
	b. No	3

Results of the Survey

* (5/5) means 5 out of 5 respondents

1. USE CASE DIAGRAM

QUESTION	YES	NO
Understandable ?	5/5	–
Useful ?	5/5	–

2. CLUSTER CHART

QUESTION	YES	NO
Understandable ?	1/5	4/5
Useful ?	1/5	4/5

3. CLASS DIAGRAM

QUESTION	YES	NO	NO ANSWER
Understandable ?	4/5	1/5	-
Useful ?	3/5	1/5	1/5

4. STATIC ARCHITECTURE

QUESTION	YES	NO	NO ANSWER
Understandable ?	2/5	3/5	-
Useful ?	3/5	1/5	1/5

5. COMPARISON: CLASS DIAGRAM AND STATIC ARCHITECTURE

QUESTION	CLASS DIAGRAM	STATIC ARCHI	NO PREFERENCE	NONE OF THEM
Preference	5/5	–	–	–

APPENDIX Q: RESULTS OF PHASE 2 AND PHASE 3 OF THE COMPARISON

A. BON'S FEATURES

1. Concepts

Main Concepts

NO	FEATURE	DESCRIPTION	ADDITIONAL INFORMATION
1.	Root class	A class of which one instance will be created when an OO process is started, and whose initialization routine drives the execution	
2.	Simplicity	A principle to minimize the number of concepts	
3.	Software contracting	A view of software development as a series of documented contracting decisions. The idea is to use assertions (pre-conditions, post-conditions and invariants) to define the semantics of each class.	• UML does not talk much about assertions, but according to Fowler (1997, 73) one can use them in UML without any trouble. • UML defines no strict syntax for describing constraints other than putting them inside braces ({}).

Relationships

NO	FEATURE	DESCRIPTION	ADDITIONAL INFORMATION
1.	Role multiplicity	The number of client relations from a client class to a supplier class.	
2.	Shared association	A special case of association in which whenever an instance of the client class is attached to an instance of the supplier class, it will always be to the same supplier instance (or one in a fixed set).	

Communications

2. Modeling Language and Techniques

NO	PURPOSE	FEATURE	ADDITIONAL INFORMATION
1.	To identify the general description of the system	System chart	
2.	To identify scenarios	Scenario chart	
3.	To identify candidate classes and their characteristics	Class chart	
4.	To identify grouping of the classes	Cluster chart	
5.	To identify external and internal events of the system	Event chart	
6.	To identify dependencies among classes	Object creation chart	
7.	To provide a sorted list of all classes in the system along with short descriptions	Class dictionary	
8.	To show role multiplicity	1. Attaching several labels to the same graphical link. 2. Using multiplicity marker (small lozenge containing the number of links)	
9.	To show shared association	Placing a special sharing marker (small circle containing the number of suppliers involved) on a client link.	
10.	To work out software contracting	Employs typed notation for each class containing: 1. assertions 2. pre-conditions 3. post-conditions 4. invariants	• Even though UML does not particularly address assertions, we can still implement them in UML. • Constraint rules on class diagram are actually equivalent to invariants • Operation pre-conditions and post-conditions may be documented within the operation definition.

B. UML'S FEATURES

1. Concepts

Main Concepts

NO	FEATURE	DESCRIPTION	ADDITIONAL INFORMATION
1.	Concurrency	A mechanism to handle concurrent behavior.	UML provides concurrent state diagram to show the behavior of a single object on concurrent events.
2.	Scenario	An instance of a use case (one path through the flow of events for the use case).	BON also has a term "scenario" which is more similat to use case in UML.

Relationships

NO	FEATURE	DESCRIPTION	ADDITIONAL INFORMATION
1.	Composition	• A stronger variety of aggregation in which the part object may belong to only one whole and the parts are usually expected to live and die with the whole. • Any deletion of the whole is considered to cascade to the parts.	
2.	Derived elements (derived association and derived attribute)	Is an element (either association or attribute) that can be calculated from another one on a class diagram.	
3.	Instance multiplicity	The number of objects (instances of a given class) that may be attached to	• BON does not provide a specific notation for

		another object through a relationship.	representing instance multiplicity. • In BON, a constraint on the number of instances of a certain class is assumed as irrelevant information at the system level, and therefore should be placed in the class interface specifications as part of the software contracts.

Communications

—

2. Modeling Language and Techniques

NO	PURPOSE	FEATURE	ADDITIONAL INFORMATION
1.	To show graphical representation of some or all of the actors, use cases, and the relationships between the use cases and the actors	Use case diagram	
2.	To show packages of classes and the dependencies among them	Package diagram	This diagram is actually just a form of class diagram, which is given different name by Fowler (1997, 114).
3.	To show object behavior within control structure	Activity diagram	
4.	To show object behavior across many use cases	State diagram	Use only for classes that exhibit interesting behaviors
5.	To show the behavior of a single object on concurrent events	Concurrent state diagram	BON does not provide any notation to show concurrency. Instead, it encourages users to employ complementary notation.
6.	To show dependencies among software components, including source code components, binary code components, and executable components.	Component diagram	
7.	To show deployment of software components on hardware nodes	Deployment diagram	
8.	To show instance multiplicity	Attaching the number of multiplicity on the association link	

C. COMMON FEATURES

1. Concepts

Main Concepts

NO	CONCEPTS	BON	UML	DESCRIPTIONS	DISCREPANCIES
1.	Attribute	Attribute	Attribute	A property of an object manifested as a function returning value.	
2.	Class	Class	Class	A description of certain kinds of objects.	
3.	Cluster	Cluster	Package	A group of model elements, such as classes, group of classes, etc.	
4.	Deferred class	Deferred class	Abstract class	• A class containing at least one feature which has no implementation and never will have. • A deferred class has no instances and is only used for classification and interface definition purposes.	
5.	Deferred features	Deferred feature	No specific term	• A feature that has no implementation (and never will have). • It serves as a (full or partial) specification of all implemented versions	

				of the feature that may occur in descendant classes.	
6.	Effective class	Effective class	No specific term	• A class all of whose features are effective. • An effective class is implementing the interface of a deferred class or reimplementing the interface of an ancestor.	
7.	Effective feature	Effective feature	No specific term	A feature which has an implementation attached to its specification.	
8.	Feature	Feature	Attributes and operations	Definition in BON: an operation applicable to an object. It covers the concept of class attribute at implementation time (state variable).	BON doesn't pay much attention on separating attributes and operations.
9.	Generic class	Generic class/ parameterized class	Parameterized class	A class from which a number of related types may be derived, depending on a number of furnished type parameters.	
10.	Information hiding	Information hiding/visibility	Visibility	A mechanism to specify whether an operation is available to a client or not.	
11.	Interface	Interface class	Interface	A class with no implementation	

12.	Method (of a class)	A class feature (aside from attribute)	Method	The body of procedure	UML differentiate method call/declaration as operation, and method body as method.
13.	Object	Object/class instance	Object/class instance	The basic components of an OO system, exhibiting well-defined behavior in response to a number of applicable operations.	
14.	Operation	Feature call	Operation	• The process that a class knows to carry out. • Common term: method call or method declaration	
15.	Persistent object	Persistent object	Persistent object	An object whose life span is independent of any system session. It may reside in either primary or secondary storage.	
16.	Refinement	Refinement	Refinement	A general term to indicate a greater level of detail. It can be used for implementation of interfaces or for some other purposes.	
17.	Reusability	Reusability	Reusability	The ability of software elements to serve for the construction of many different applications.	

18.	Reversibility	Reversibility	Reversibility	The possibility of seamlessly translating changes made during a certain development phase back into earlier phases, so as to maintain consistency.	
19.	Scalability	Scalability	Scalability	The ability of a method and notation to scale up from small textbook examples to sizeable real applications without losing usability	
20.	Scenario	Scenario	Use case	A script of a possible system execution showing the objects involved, which other objects they call, and the temporal order of these calls.	• The term scenario in BON is the same with use case in UML. • However, in UML there is a term called scenario which means an instance of a use case (one path through the flow of events for the use case).
21.	Seamlessness	Seamlessness	Seamlessness	The quality of a natural translation from problem domain specification, over system design, into executable code.	

Relationships

NO	RELATIONSHIP	BON	UML	DESCRIPTIONS	DISCREPANCIES
1.	Aggregation	Aggregation	Aggregation	A relationship representing a composition of a group of objects into an integral unit (a part-of relationship)	
2.	Association	Association	Association	A relationship denoting a logical dependency between two classes.	In BON, association and aggregation relationships are subtypes of client relation.
3.	Inheritance	Inheritance	Inheritance	A relationship among classes where one class shares the structure and/or behavior defined in one or more other classes.	
4.	Multiple inheritance	Multiple inheritance	Multiple inheritance	A mechanism allowing one class to inherit simultaneously from more then one parent class.	
5.	Redefinition	Redefinition	Overriding	A modification in a descendant class of the implementation of an inherited feature.	
6.	Repeated inheritance	Repeated inheritance	Repeated inheritance	• The same feature being inherited more than once from a	

				class. • Repeated inheritance in BON can be divided into two types: 1. Direct repeated inheritance 2. Indirect repeated inheritance	

Communications

NO	COMMUNICATION	BON	UML	DESCRIPTIONS	DISCREPANCIES
1.	Polymorphism/dynamic binding	Polymorphism	Polymorphism	• A mechanism to provide capability for clients to manipulate objects in terms of their common superclasses.	

2. **Modeling Language & Techniques**

NO	PURPOSES	BON	UML	DISCREPANCIES
1.	To show static structure of the classes	Static architecture	Class diagram	
2.	To show groups of classes and dependencies among them	Static architecture.	Package diagram	
3.	To show object collaboration (in a single use case/scenario)	Object scenario	Interaction diagram: • Sequence diagram: time based order • Collaboration diagram: organized around the object links to one another	
4.	To show association relationship	Association link	Association	• In UML, associations in a class diagram may be represented with *navigabilty* (represented by an arrow) indicating the responsibility of on object towards another. • In UML, an association without arrows means either the navigability is unknown or the association is bi-directional.

				• In BON, an association link is always ended with an arrowhead (or an open brace in case of aggregation) pointing to the supplier
5.	To show aggregation relationship	Aggregation link	Aggregation	
6.	To show inheritance relationship	Inheritance link	Generalization path	• BON provides simplified inheritance link for direct repeated inheritance by using a marker (small lozenge containing the number of inheritances).
7.	To achieve weak/low coupling	1. Keep dependencies bet-ween classes in different clusters to a minimum 2. Restricting cycles between classes (where both classes use facilities from each other) to occur only within the same clusters.	1. Keeping dependencies to a minimum. 2. Remove cycles in dependency structures → employ package generalization.	

8.	To achieve scalability	Employs: 1. Element compression and expansion 2. Nested clustering	Employs: 1. Package diagram 2. Deployment diagram	
9.	To express genericity	Generic class/parameterized class	Parameterized class	
10.	To implement information hiding	Implements: 1. Public feature: visible to any client class 2. Restricted feature: available only to a certain group of other classes.	Implements: 1. Public member: visible anywhere in the program and may be called by any object within the system 2. Private member: may be used only by the class that defines it. 3. Protected member: may be used only by the class that defines it and a subclass of that class.	In BON, restricted feature also includes private feature which is not visible to any other classes.

3. Tools and users

NO	CRITERIA	BON	UML	DESCRIPTION
1.	CASE tools supporting the method	EiffelCase®	• Rational Rose® • Object Team®	
2.	Number of projects that have been done using the method.	43	No information available	
3.	Number of companies supporting the method.	No information available	18	

BIBLIOGRAPHY

BOOKS, JOURNALS AND MAGAZINES

Agnes, Michael, chief ed. 1996. Webster's New World Dictionary and Terms. New York: MacMillan.

Arnold, P., S. Bodoff, D. Coleman, H. Gilchrist, and F. Hayes. 1991. An Evolution of Five Object-Oriented Development Methods. Research Report HP Laboratories. June.

Berard, Edward V. 1993. Essays on Object-Oriented Software Engineering. Volume I. New Jersey: Prentice-Hall, Inc.

Booch, Grady, Peter Coad, Meilir Page-Jones, Paul Ward, Kent Back. 1991. Can Structured Methods be Objectified ? In OOPSLA'91 Conference Proceedings. New York: ACM Press.

Booch, Grady. 1994. Object-Oriented Analysis and Design with Applications. 2nd ed. California: the Benjamin/Cummings Publishing Company, Inc.

Coleman, Derek, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilchrist, Fiona Hayes, and Paul Jeremaes. 1994. Object-Oriented Development : The Fusion Method. New Jersey: Prentice-Hall, Inc.

Cribbs, J., C. Roe, and S. Moon. An Evaluation of Object-Oriented Analysis and Design Methodologies. SIGS book.

De Champeaux, Dennis and P. Faure. 1992. A Comparative Study of Object-Oriented Analysis Methods. Journal of Object-Oriented Programming. Vol. 5, No.1. (March/April): 21-32.

Eliëns, Anton. 1995. Principles of Object-Oriented Software Development. Cambridge: Addison-Wesley Publishing Company.

Fichman, Robert G. and Chris F. Kemerer. 1992a. OO and Conventional Analysis and Design Methodologies. Computer. Vol. 25, No. 10. October. 22-40.

Fichman, Robert G. and Chris F. Kemerer. 1992b. Object -Oriented and Conventional Analysis and Design Methods - Comparison and Critique. Computer. October. 22-39.

Fichman, Robert G. and Chris F. Kemerer. 1997. Object Technology and Reuse: Lessons from Early Adopters. Computer. Vol. 30, No. 10. October.

Fowler, M. 1994. Describing and Comparing Object-Oriented Analysis and Design Methods. Object Development Method. Edited by A. Carmichael. SIGS book.

Fowler, M. 1997. UML Distilled: Applying the Standard Object Modeling Language. Reading, Massachusetts: Addison-Wesley.

Gore, Jacob. 1996. Object Structures: Building Object-Oriented Software Components with Eiffel. Reading, Massachusetts: Addison-Wesley Publishing Company Inc.

Graham, Ian. 1994. Object-Oriented Methods. 2nd ed. London: Addison-Wesley Publishers Ltd.

Harmon, Paul, and William Morrisey. 1996. The Object Technology Book. New York: John Willey & Sons Inc.

Hazeltine, Nelson, Tim Hilgenberg, Reed Philips, David Taylor, Timothy Korson. 1991. Managing the Transition to Object-Oriented Technology. In OOPSLA'91 Conference Proceedings. New York: ACM Press.

Jacobson, Ivar. 1992. Object-Oriented Software Engineering, a Use Case Driven Approach. New York: Addison-Wesley.

Lewis, John A. Sallie M. Henry, Dennis G. Kafura, Robert S. Schulman. 1991. An Empirical Study of the Object-Oriented, Paradigm and Software Reuse. In OOPSLA'91 Conference Proceedings, New York: ACM Press.

Lorenz, Mark. 1993. Object-Oriented Software Development. New Jersey: Prentice-Hall Inc.

Malan, Ruth, Derek Coleman, Reed Letsinger. 1995. Lessons from the Experiences of Leading-Edge Object Technology Projects in Hewlett-Packard. In OOPSLA'95 Conference Proceedings, New York: ACM Press.

Martin, James, and James J. Odell. 1994. Object-Oriented Analysis & Design. New Jersey: Prentice-Hall.

Martin, Robert C. 1995. Designing Object-Oriented C++ Applications Using the Booch Method. New Jersey: Prentice-Hall, Inc.

Meyer, Bertrand. 1997a. Object-Oriented Software Construction. 2nd ed. New Jersey: Prentice-Hall PTR.

Meyer, Bertrand. 1997b. The View from the Eiffel Tower. Interview by Carlo Pescio. Software Development. (September): 51-6.

Monarchi, David, and Gretchen I. Puhr. 1992. A Research Typology for Object-Oriented Analysis and Design. Communication of the ACM. Vol. 35, No. 9. September. 35.

Nerson, Jean-Marc. 1992. Applying Object-Oriented Analysis and Design. Communications of the ACM. Vol. 35, No. 9. September. 63-74.

Novak Jr, G.S. 1997. Software Reuse by Specialization of Generic Procedures through Views. IEEE Transactions on Software Engineering. Vol. 23, No. 7. July.

Pfaffenber, Bryan. 1992. Que's Computer Dictionary. 3rd ed. Indiana: Que Corporation.

Pfleeger, Shari Lawrence. 1992. Software Engineering: The Production of Quality Software. 2nd edition. Philippine Reprint. Quezon City: St. Martin Publications.

Quatrani, Terry. 1998. Visual Modeling with Rational Rose and UML. Reading, Massachusetts: Addison-Wesley.

Rine, David C. 1997. Supporting Reuse with Object Technology. Computer. Vol. 30, No. 10. October.

Rubin, Kenny, John Daniels, Charles Berman, James Coplien, Dough Johnson, Laura Hill. 1995. Managing Object Oriented Projects. In OOPSLA'95 Conference Proceedings. New York: ACM Press.

Rumbaugh, James. 1997. Modeling through the Years. Journal of Object-Oriented Programming (JOOP). (July/August): 16-19.

Taylor, David A. 1990. Object-Oriented Technology. California: Addison-Wesley Publishing Company.

Tkach, Daniel and Richard Puttick. 1996. Object Technology in Application Development. 2nd ed. Reading, Massachusetts: Addison-Wesley Publishing Company.

Waldén, Kim and Jean-Marc Nerson. 1995. Seamless Object-Oriented Software Architecture. Great Britain: Prentice-Hall.

Whitten, Jeffrey L., Lonnie D. Bentley, and Victor M. Barlow. 1994. Systems Analysis and Design Method. 3rd ed. Illinois: Irwin.

Wirfs-Brock, Rebecca J., Ralph E. Johnson. 1990. Surveying Current Research in Object-Oriented Design. Communication of the ACM. Vol. 33, No. 9. September. 104-124.

WEB SITES

Anonymous. BON: The Analysis and Design Method for Reliability, Reusability and Reversibility. http://www.eiffel.com/products/bon.html. Accessed 16 February 1998.

Anonymous. Object-Oriented FAQ Section 1:Basics. http://www.cyberdyne-object-sys.com/oofaq2/basic.html. Accessed 20 February 1998.

Anonymous. Object-Oriented FAQ Section 3:General. http://www.cyberdyne-object-sys.com/oofaq2/general.html. Accessed 20 February 1998.

Anonymous. Eiffel Homepages. http://www.eiffel.com. Accessed 2 September 1998.

Anonymous. UML Homepages. http://www.rational.com/uml. Accessed 19 February 1998.

Anonymous. UML FAQ. http://www.rational.com/uml/faq.html. Accessed 6 January 1999.

Anonymous. UML Summary. http://www.rational.com/uml/summary.html. Accessed 19 February 1998.

Anonymous. UML Notations. http://www.rational.com/uml/html/notation/. Accessed 19 February 1998.

Anonymous. UML Semantics. http://www.rational.com/uml/html/semantics/. Accessed 19 February 1998.

Anonymous. Rational Unified Process: Best Practices for Software Development Team. http://www.rational.com/uml/whitepapers. Accessed 27 December 1998.

Anonymous. ISE Eiffel discussion: EiffelCase and BON. http://eiffel.com/services/userlist//eiffelcase_and_bon.html. Accessed 25 February 1999.

Anonymous. ISE Eiffel discussion: AADRE: Recent messages. http://eiffel.com/services/userlist//aadrecent_messages.html. Accessed 25 February 1999.

Anonymous. Some ISE Eiffel projects. http://eiffel.com/eiffel/projects/list.html. Accessed 2 September 1998.

Ashrafuzzaman, Mohammad. Object-Oriented Analysis and Design. http://www.cs.usask.ca/homepages/grads/moa135/856/OO/node4.html. Accessed 13 February 1998.

Brinkkemper, Sjaak, Shuguang Hong, Arjan Bulthuis, Geert van den Goor. Object-Oriented Analysis and Design Methods, a Comparative Review. http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html. Accessed 20 February 1998.

Demmer, Christian. Unified Modeling Language vs. MWOOD-I. http://stud2.tuwien.ac.at/~e8726711/ummw2#MU3. Accessed 20 February 1998.

Hong, Shuguang, Geert van den Goor, Sjaak Brinkkemper. A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In The Proceedings of the 26th Hawaii International Conferences on System Sciences, Volume IV, January 1993. http://cis.gsu.edu/~shong. Accessed 19 January 1998.

Jezequel, Jean-Marc. Object-Oriented Software Engineering with Eiffel. http://www.irisa.fr/pampa/EPEE/book. Accessed 19 February 1998.