OBJECT-ORIENTED METHODOLOGIES: A MATTER OF PERCEPTION!

John Pugh and Amir Zeid

TR-254 OCTOBER 1994

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

Object-Oriented Methodologies: A Matter of Perception![*]

John Pugh[†] Amir Zeid[†]

Abstract

Object-oriented analysis and design is a rapidly expanding field. There are about 20 methods claiming to support full development lifecycle. Several methods cannot be judged easily, because they are not well documented, are quite immature, are a single short publication, or are not intended for the open market. The existing methodologies change dramatically from year to year. Now, we reached a crucial point in methods evolution. New methods claim to represent a second-generation. Which methodology is better? Is choosing a method independent from the problem or not? How to choose an appropriate methodology? Are the second-generation of methods better than the first-generation? Are we on the right track of evolution? Do we need more methods? What should we look for next? Does the perfect methodology exist or there is always a missing piece of the puzzle? If there is something missing in a method, where is it? The answers of these questions are debatable, and it is always a matter of perception!

In this paper, we try to answer these questions for a subset of the methodologies. We compare among three different first-generation methodologies, which are OMT by Rumbaugh, OOD by Booch and RDD by Wirfs-Brock, and the first second-generation method which is Fusion, by solving the same problem using each methodology.We also include a study for the four methodologies in details. Then we try to combine the best of the methodologies in a coherent way.

Key Words and Phrases: Object-oriented Methodologies, Analysis, Design.
Carleton University, School of Computer Science: SCS-TR-254

---

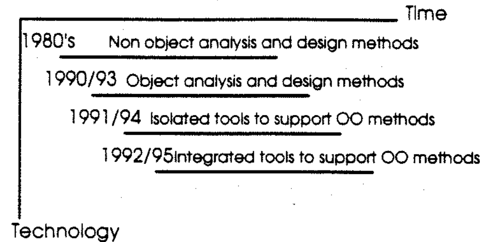
[*] Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.
[†] School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada. Email: {Pugh, Zeid}@scs.Carleton.ca

1 Introduction

An analysis or design method is a coherent approach to describing a system. This system may be a computer system, or it may be the business process that a computer system is going to support [6]. An object-oriented methodology is a method that supports object-oriented concepts such as inheritance, polymorphism and encapsulation. Methods use different techniques to describe the system, usually techniques are not exclusive to one method and the same technique may appear in a range of methods.

1.1 Evolution of object-oriented methods

Object analysis and design is an evolution of the analysis and design methods which were developed in the 1970's and 80's. Most of the issues which were addressed by those methods are also relevant in object analysis and design methods. Now the demand for object-oriented approaches is staggering. The promises of object-oriented technology are being heard in places where methodologies are the norm. Further, it is perfectly normal to someone to say: "I want to do object-oriented software engineering. How do I do it and more importantly how do I know that I have done it well?"
The diagram below, shows the road map of how object analysis and design is expected to grow to widespread use.



1.2 Why object-oriented methodologies?

Object-oriented methodologies bring several benefits:
(i)A consistent modeling approach.
(ii) The ability to reuse specifications.
(iii)A clearer, more natural means of communication between analyst and the user.
Also, object-oriented methods aim at solving the existing problems with object orientation. The most well-known problems are:
-Focus on code: The common emphasis in object orientation is on the programming techniques and languages, not on the development process. When expressed in programming terms, analysis and design models are not abstract enough.
-Team working not addressed: Software is usually developed by teams, not individuals. Object orientation provides little help for this.
-Difficult to find objects: Finding the right objects and classes in an object-oriented system is not easy. This is partly due to the unfamiliarity of the approach, but it is also difficult in itself.
-Function-oriented methods are inappropriate: When object orientation is introduced at the programming language level in a software development environment, the traditional methods of analysis and design no longer work!
Solving these problems is not just a matter of more powerful programming languages, or better programmer education. What is needed are development processes specific for object-oriented software.

1.3 Why OMT, OOD, RDD, Fusion?

We chose these four methodologies because each one of them represents a different approach. OMT's main concern is modeling and it applies some techniques which are already found in structured analysis, OOD is full of graphical notations and it views the system in a different way using logical and physical views, while RDD is totally different because it describes the classes in terms of their responsibilities and it abandons data modeling. Finally, Fusion claims to be the first second-generation object-oriented method, and by testing it we should be able to determine if object-oriented methods evolution is on the right track or not.
Methods also can be classified as revolutionary and evolutionary. OMT and Fusion can be classified as evolutionary, while OOD and RDD can be classified as revolutionary since they bring new ideas, so by this comparison we can determine also which approach is more successful.

1.4 Problem statement

We will use the following problem throughout this paper as our example:
We are required to design the software to support a computerized banking network including both human cashiers and automatic teller machines (ATMs) to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them. Cashier stations are owned by individual banks and communicate directly with their own bank's computers. Human cashiers enter account and transaction data. Automatic teller machines communicate with a central computer which clears transactions with the appropriate banks. An automatic teller machine accepts a cash card, interacts with the user, communicates with the central system to carry out the transaction, dispense cash, and prints receipts. The system requires appropriate record keeping and security provisions. The system must handle concurrent accesses to the same account correctly. The banks will provide their own software for the ATMs and the network. The cost of the shared system will be apportioned to the banks according to the number of customers with cash cards [6].

2 Solving the problem using Rumbaugh's Method (OMT)*

2.1 The method

OMT is a methodology designed by Rumbaugh, OMT stands for Object Modeling Technique. For a subset of the notations used in OMT, refer to [7]. The lifecycle of the method is unnamed in the reference, but it can be seen as cascading waterfall. The method consists of four phases:
(i) Analysis, during which the object model (a static model for the classes and their relationships + data dictionary), and the dynamic model (state diagrams for classes + global event flow diagram + event trace diagrams) and finally a functional model (data flow diagrams) are constructed. The purpose of the analysis phase is to model the real world so that it can be understood.
(ii) System design, during which the basic architecture of the system is defined and the system is partitioned into subsystems and also to processors and tasks.
(iii) Object design, during which objects are defined in details (interface, algorithm and operation). The objects discovered during analysis serve as skeleton for this activity.
(iv) Implementation, during which the implementation of the defined objects is done. OMT provides guidelines for using both object-oriented and non-object-oriented languages.

2.2 The deliverables

The deliverables of the method are analysis document which consists of the three models, and design document which includes a more detailed version of the models.

---

* Figures are included in Appendix A, part 1.

2.3 The solution

The following solution was done using OMTool** which is a CASE tool that supports OMT.

(i)Object modeling

Now we will start applying the method, first during analysis we have to define the data dictionary then from it draw the object model diagram, these two components constitute the first deliverable. So let us start by the data dictionary:

A-Data dictionary

Account-a single account in bank against which transactions can be applied. Account may be of various types, at least checking or savings. A customer may hold more than one account.

ATM-a station that allows customers to enter their own transactions using cash cards as identification. The ATM interacts with customer to gather transaction information, sends the transaction information to the central computer for validation and processing, and dispenses cash to the user.

Bank-a financial institution that holds accounts for customers and that issues cash cards authorizing access to accounts over the ATM network.

Bank computer-the computer owned by the bank that interacts with the ATM network and the bank's own cashier stations.

Cash card-a card assigned to a bank customer that authorizes access of accounts using an ATM machine. Each card contains a bank code and a card number. The bank code uniquely identifies the bank within the consortium. The card number determines the accounts that the card can access. Each cash card is owned by only one customer.

Cashier-an employee of a bank who is authorized to enter transactions into cashier stations and accept and dispense cash and checks to customers. Transactions, cash, and checks handled by each cashier must be logged and properly accounted for.

Cashier station-a station on which cashiers enter transactions for customers. Cashiers dispense and accept cash and checks; the station prints receipts. The cashier station communicates with the bank computer to validate and process the transactions.

Central computer-a computer operated by the consortium which dispatches transactions between the ATMs and the bank computers. The central computer validates bank codes but does not process transactions directly.

Consortium-an organization of banks that commissions and operates the ATM network. The network only handles transactions for banks in the consortium.

Customer-the holder of one or more accounts in a bank. A customer can consist of one or more persons. The same person holding an account at a different bank is considered a different customer.

Transaction-a single integral request for operations on the accounts of a single customer. We only specified that ATMs must dispense cash, but we should not preclude the possibility of printing checks or accepting cash or checks. We may also want to provide the flexibility to operate on accounts of different customers, although it is not required yet.

B-Object model diagram

The object model describes the structure of objects in a system-their identity, their relationships to other objects, their attributes, and their operations. The nodes of this diagram are the classes and the arcs are the relationships among them. The relationships may be inheritance, association or others. Figure 1 shows the object model with attributes and inheritance.

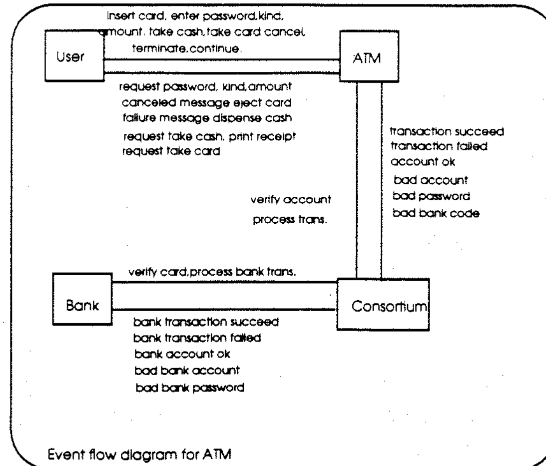
(ii)Dynamic modeling

The dynamic model shows the time dependent behavior of the system. It consists mainly of a general event flow diagram, event trace diagrams to trace the scenarios and state transition diagrams for the classes.

** Solution was done using OMTool ver 2.0 for PCs.

A-Event flow diagram

The following figure shows an event flow diagram for the ATM system. The diagram summarizes events between classes, without regard for sequences. Include events from all scenarios, including error events. The event flow diagram is a dynamic counterpart to an object diagram. Paths in the object diagram shows possible information flows, while paths in the event flow diagram shows possible control flows.



Event flow diagram for ATM

B-The event trace diagram

The sequence of events and the objects exchanging events are described in an event trace diagram. This diagram shows each object as a vertical line and each event as a horizontal arrow from the sender to the receiver. Time increases from top to bottom. The following scenario will be used as our example in this paper:

1-The ATM asks the user to insert a card; the user inserts a cash card.

2-The ATM accepts the card and reads its serial number.

3-The ATM requests the password; the user enters it.

4-The ATM verifies the serial number and password with the consortium; the consortium checks it with bank and notifies the ATM of acceptance.

5-The ATM asks the user to select the kind of transaction; the user selects withdrawal.

6-The ATM asks for the amount of cash; the user enters $100.

7-The ATM verifies that the amount is within predefined policy limits and asks the consortium to process the transaction; the consortium passes the request to the bank, which eventually confirms success and returns the new account balance.

8-The ATM dispenses cash and asks the user to take it; the user takes the cash.

9-The ATM asks whether the user wants to continue; the user indicates no.

10-The ATM prints a receipt, ejects the card, and asks the user to take them; the user takes the receipt and the card.

10-The ATM asks a user to insert a card.

Figure 2 shows the event trace diagram for this scenario.

C-The sate transition diagrams

A state diagram relates events and states. When an event is received, the next state depends on the current state as well as the event; a change of state caused by an event is called a transition. The nodes of the diagram are states while the arcs are transitions. Figures 3,4,5 show examples of state transition diagrams for major classes in the ATM system.

(iii)Functional modeling

The functional model shows how results are computed, without regard for sequencing, decisions, or object structure. It consists mainly of data flow diagrams. Figures 6,7 show two examples of the data flow diagrams: a general overview and a transaction process.

The last deliverable is done during the design phase which is a system architecture diagram which shows the major subsystems. Figure 8 shows the architecture of the ATM system.

2.4 Comments about Rumbaugh's approach

(i)OMT is affected by relational database design and several concepts essential for such work are introduced.

(ii)OMT uses techniques which are already found in structured analysis and puts them together well. Those used to structured approaches may find these methods familiar and easy to learn, but this makes the method a hybrid one (not a pure object-oriented methodology).

(iii)OMT gives a large amount of good and explicit advice that is needed in common situations which is very helpful for designers.

(iv)The primary strength of OMT is its analysis phase. It has a well defined process and all the models use concise and understandable notations. The relationship between the three models is not clear. They are developed more or less independently, and it can be difficult to get the overall picture.

(v)The main weakness is that the design stage lacks the step-by-step approach of the analysis phase. It is less clear where to start and what to do next, but it does embody some useful heuristics.

(vi)Comments about OMTool (CASE tool that supports OMT).

A-OMTool is not user friendly, every thing is done through menus. No icons are provided which makes it a bit difficult. It should have included both facilities.

B-Switching between the diagrams has to be done sequentially! This is a real waste of time if you are working with a large system with 20 diagrams for example. It should have provided a pane for the diagrams to choose from.

C-Adding a new class is a problem. You have first to create it, fill its template then add it to the image, otherwise it will not be added and it just exists graphically not logically.

D-OMTool does not have an editor for event traces or event flows, or the architecture. It just provides facilities for Object, Dynamic, Functional modeling.

E-There are some differences in notations between OMTool and the method itself which creates confusion. A starting state in the state transition diagrams is a good example of inconsistency between the tool and the method. The difference is clear when you compare the figures generated by OMTool (Figures 3,4,5) and the actual notations described in Rumbaugh's book (Appendix A).

F-OMTool does not provide a facility to build a data dictionary explicitly, instead it has a nice browser to give the user the facility to look at the classes.

G-OMTool provides code generation facility in C++.

H-OMTool has no on-line help, no support for multiple drives, and the system is obviously a poor port from UNIX.

3 Solving the problem using Booch's method (OOD)*

3.1 The method

OOD is a methodology designed by Booch, OOD stands for Object Oriented Design. For a subset of the notations used in OOD refer to [1].

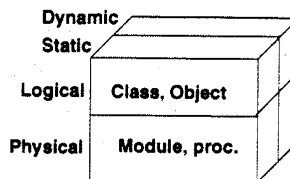
The method supports iterative development and staged, incremental prototypes that lead to final

* Figures are included in Appendix A, part 2.

product. The method consists mainly of four phases:
(i)Finding classes and objects, during this phase key abstractions are defined. These key abstractions are found by learning the terminology of the problem domain.
(ii)Identifying the semantics of the classes, during this phase the meanings of the classes and objects identified earlier are defined. To investigate how each object may be used by other objects is also an essential part of identifying its semantics.
(iii)Identifying relationships between classes, during this phase the previous activities are extended to include also the relationships between classes and objects. Different types of associations are defined during this phase.
(iv)Implementing classes and objects, during this phase a decision is made of how to use a particular programming language to implement the classes [6].

3.2 The deliverables

The deliverables of this method are the logical and physical view of the system. Each of which has a static and dynamic dimension.



The static dimension of the logical view is the relationships between classes or the class diagrams, while the dynamic dimension is the object timing diagrams which show some major use-cases and the state transition diagrams for each class. As for the physical view, it consists mainly of the modules and the processes diagrams. The following figure shows how Booch views the system.

3.3 The solution

The following solution was done using Rose** which is a CASE tool that supports OOD.

(i)Logical view.

Now we will start applying the method

A-Class diagram

The class diagram shows the existence of classes and their relationships in the logical view of a system. Figure 9 displays the class diagram of the ATM system.

B-State transition diagrams

State transition diagrams show the state space of a given class, the events that cause a transition from one state to another, and the actions that result from a state change. OOD uses Harel state charts which is the same concept used in OMT, but with a slight difference in notations. Figures 10,11 show examples of two major classes.

C-Object timing diagram

Object timing diagram represents a snapshot in time of an otherwise transitory stream of events over a certain configuration of objects. Each object diagram represents the interactions or structural relationships that may occur among a given set of class instances [1].
We will use the same scenario used in the OMT solution as our use-case. Figure 12 shows an object timing diagram of the scenario.

** Solution was done using Rose ver 1.3 for PCs.

(ii)Physical view

A-Module diagrams

Module diagrams show the allocation of classes and objects to modules in the physical view of a system. The concept of modules does not exist in pure object-oriented languages, so it will be ignored here since we are aiming at programming using a real pure object-oriented language like Smalltalk.

B-Process diagram

The process diagram shows the allocation of processes to processors in the physical view of a system. A single process diagram represents a view into the process structure of a system. Figure 13 is the process diagram for the ATM system.

3.4 Comments about Booch's approach

(i)OOD uses awkward notation figures. They are very difficult to be drawn manually or even using a computerized graphics package, so it is essential to have a tool that supports the notation or a long time will be wasted just for drawing.

(ii)OOD's major strength is the richness and variety of the diagramming techniques it offers. As illustrated in the example it has at least five different diagramming techniques This variety gives a lot of flexibility, particularly when dealing with concurrent systems; however, the notation is complex and information can be fragmented and duplicated across models. So we can say that its notations are too rich!

(iii)OOD's main weakness is the absence of a defined process for developing systems.

(iv)OOD uses the "round-trip gestalt" style which emphasizes the incremental and iterative development of a system through the refinement of different yet consistent views of the system as a whole.

(v)Comments about Rose(CASE tool that supports OOD).

A-Rose has a user friendly interface, it has both menus and graphical interface.

B-Rose provides tools for almost all the diagrams described in the method.

C-Rose does not provide most of the advanced features of OOD. It just provides all the tools but with minimum options. Many examples can be shown to prove this point. We will give one example for each diagram just for the sake of proving our point: Abstract classes can not be differentiated from other classes(class diagram), the direction of data flow can not be shown(object timing diagram), keeping track of history is not provided(state transition diagram).

D-Rose uses different notations from those described in OOD for state transition diagrams which creates inconsistency between the tool and the method, refer to Appendix A and figures 10,11 to see the difference.

E-Rose does not provide code generation facility; however, the latest version of Rose does provide code generation facilities.

4 Solving the problem using Wirfs-Brock method (RDD)*

4.1 The method

RDD is a methodology designed by Wirfs-Brock, RDD stands for Responsibility Driven Design. For a subset of the notations used in RDD refer to [8].

The method supports spiral development (iterate through requirements specification, design, implementation and testing). The method consists mainly of two phases:

(i)The exploratory phase, during which the designer is responsible to find the classes of the system,

* Figures are included in Appendix A, part 3.

their responsibilities and how they collaborate with other classes. This information is normally written in what is called CRC cards (Class-Responsibility-Collaborators). The classes are extracted by reading the specification and extracting the essential nouns and then the purpose of each class is defined. Responsibilities are defined by looking for verbs in the specification. Collaborations are defined by asking questions like 'What needs to make use of this class?'

(ii) The refinement phase, during which the designer is responsible to define the inheritance hierarchies and the subsystems, if any exists. By using Venn diagrams, abstract classes can be extracted. The collaborations diagrams can then be defined.

4.2 The deliverables

The deliverables of this method are:

(i) Specification of each class(CRC cards).

(ii) Class hierarchy diagram.

(iii) Collaborations diagrams.

4.3 The solution

(i)Exploratory phase.

Now we will start applying the method, during the exploratory phase CRC cards have to be defined. Responsibilities include two items: the knowledge an object maintains and the actions an object can do. Responsibilities are meant to convey a sense of the purpose of an object and its place in the system. Collaborations represent requests from a client to a server in fulfillment of a client responsibility. It is said that an object collaborates with another if, to fulfill a responsibility, it needs to send the other object any messages. Figure 16 shows CRC cards for major classes of our example.

(ii)Refinement phase

A-Class hierarchy diagrams

A hierarchy diagram is a tool that presents a graphical representation of the inheritance relationships between classes.

Figure 14 shows the hierarchy of the ATM system.

B-Collaborations diagram

A collaborations diagram displays the collaborations between classes and subsystems in graphical form. One can use the diagram to help identify areas of unnecessary complexity, duplication, or places where encapsulation is violated. Collaborations diagrams represent classes, contracts and collaborations. Contracts are the ways in which a given client can interact with a given server. Figure 15 shows the collaborations diagram for the ATM system.

4.4 Comments about the RDD approach

(i)RDD describes the system in a different way from the other methodologies. It focuses on describing the system in terms of responsibility, which is taken on by classes.

(ii)RDD abandons data modeling, it uses only simple inheritance graph. So one of the main advantages of this technique is that it emphasizes encapsulation by hiding the data structure [2]. But at the same time, getting the whole picture might be difficult since the relations between objects are not explicitly documented.

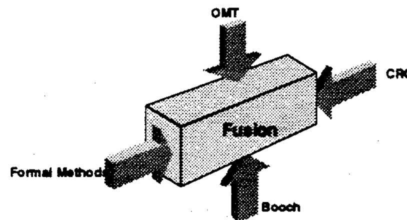
(iii)RDD lacks a systematic way to capture the dynamic behavior of objects. Instead it uses collaborations diagrams which can be seen as a union of interaction diagrams.

(iv)RDD is probably driven from a Smalltalk background because it has many fundamental similarities with Smalltalk.

(v)The big question about the RDD is whether it is possible to effectively describe systems without defining a data model. This question is open, but we think that we can not describe large systems without defining a data model.
(**vi**)RDD does not express multiple views of objects. Not all operations provided by an object are necessarily of interest to other objects that use its services. Therefore, it is desirable to define views on an object, differentiating between clients.
(**vii**)CRC cards are very useful, simple and to the point.
(**viii**)RDD expounds the benefits of using domain knowledge while preparing the user's requirement specifications. Integrating the domain knowledge with these specifications; however, can create an excessive number of objects, although only a few of these objects may be relevant to the problem being analyzed.
(**ix**)RDD is a powerful technique that is best employed as part of another method.

5 Solving the problem using Fusion*

Fusion is a methodology designed at HP Laboratories, Bristol, UK. Fusion claims to be the first child of the second-generation object-oriented methods because it borrows the best aspects of other existing methods.
It even borrows some concepts from structured analysis methods. Fusion is trying to put these concepts together and extend them so that they represent a coherent method. The following figure shows how we can view Fusion as a combination of other methods.



5.1 The method

The lifecycle of the method is similar to OMT(cascaded waterfall). The method consists mainly of three well defined phases:
(**i**)Analysis phase, during which the intended behavior of the system is defined. Models of the system are produced to describe classes and their relationships. In contrast to other methods, the analysis does not attach methods to particular classes; this is done later. During analysis two main models are defined: the object model which describes the classes and relationships and the interface model which defines the input and output communication of the system in terms of events and agents.

(**ii**)Design phase, during which the designer chooses how the system operations are to be implemented by the run-time behavior of interacting objects. During design operations are bound to their classes. During this phase four major components are defined to describe how system operations are implemented by interacting objects, how classes refer one to another. the four components are Object interaction graphs, visibility graphs, class descriptions and inheritance graphs.

---

* Figures are included in Appendix A part 4.

(iii) Implementation phase, during which the implementor runs the design into code in a particular programming language. Fusion gives guidance on how this is done [3].

5.2 The deliverables

As we have seen Fusion has many models throughout the lifecycle. The following is a summary of the deliverables in order of phases.

5.2.1 Analysis phase.

A-Object model.

B-Interface model.

1-Operation model.

2-life-cycle model.

5.2.2 Design phase.

A-Object interactions graphs.

B-Visibility graphs.

C-Class descriptions.

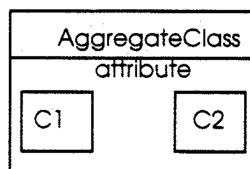
D-Inheritance graphs.

5.3 The solution

(i)Analysis phase

Two models are defined in this phase:

A-Object Model which defines the static structure of the information in the system. The purpose of this model is to capture the concepts that exist in the domain of the problem and the relationships between them. The notation is based on the entity relationship notation. It can represent classes, attributes, and relationships between classes. The extensions permit the use of aggregation and generalization. The notation is slightly different from Rumbaugh's method. Rumbaugh himself mentioned that the object model is mostly out of OMT. The main difference is in aggregation, so it is similar to figure 1.



Aggregation in Fusion

B-Interface Model which defines the input and output communication of the system in terms of events and agents. Agents model human users , or other hardware or software systems. An event is an instantaneous and atomic unit of communication between the system and the environment. This means that the interface model's main function is to capture different aspects of behavior. In order to do so it is subdivided into two models:

1-**Operation model** which characterizes the effect of each system operation in terms of the state change it causes and the output events it sends. A system operation may

-Create an instance of a class.

-Change an attribute of an existing object.

-Add or delete some tupel of objects from a relationship.

-Send an event to an agent.

The operation model is expressed as a series of schemata. There must be at least one schema for each system operation. Figure 17 is an example of operation model.

2-**life-cycle model** which characterizes the allowable sequencing of system operations and events. A lifecycle expression defines the allowable sequences of iterations that a system may participate in

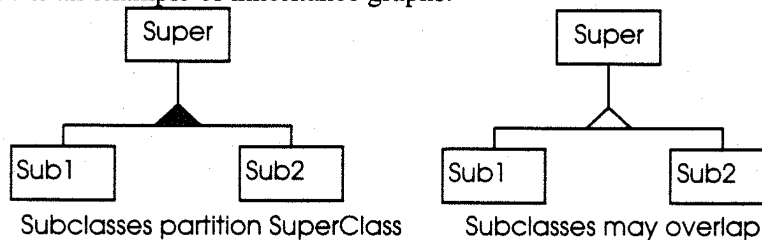
over its lifetime. life-cycle expressions have syntax and semantics and can be seen as simple extensions to regular expressions or grammars. Also, life-cycle expressions define patterns of communication.
Figure 18 is an example of lifecycle model.

(ii)Design phase
During design four major components are defined:
A-Object interaction graphs which describe how objects interact at run-time to support the functionality specified in the operation model. An object interaction graph is constructed for each system operation. Figure 19 is an example of object interaction graphs.
B-Visibility graphs which describe the object communication paths. The task of visibility graphs is to identify for each class:

-Objects the class instances need to reference.

-Appropriate kinds of reference to those objects.

These graphs are very important; however, we do not need them in our example since all objects can communicate.
C-Class descriptions which provide a specification of the class interface, data attributes, object reference attributes, and method signatures for all of the classes in the system.
D-Inheritance graphs which describe the class/subclass inheritance structure. Fusion differentiates between disjoint subtypes(subcalsses partition superclass), and nondisjoint subtypes(subclasses may overlap). Figure 20 is an example of inheritance graphs.



Subclasses partition SuperClass Subclasses may overlap

Fusion also maintains a data dictionary, a place where the different entities of the system can be named and described. This is referenced throughout the development process.

5.4 Comments about Fusion
(i) It is necessary to decide how an object knows of another's existence. Booch documents this by adding visibility information to object diagrams. In Fusion, this is documented separately and clearly on visibility graphs. Most other methods pay little attention to this aspect of design.
(ii) The book provides helpful guidelines to make use of reuse in analysis, design and implementation phases. Unfortunately, the discussion on how reuse impacts Fusion is disappointing! The lack of a deeper discussion in unfortunate given that many systems today are not built from scratch [3].
(iii) Fusion provides guidelines to combine the method with other existing methods [3]. This is very important because an immediate switch to Fusion may not be a realistic option for an organization that has already invested in another object-oriented method. In this situation, it may be possible to supplement the existing method with aspects of Fusion. The purpose of doing this would be to strengthen the existing method. This has to be done with some care to avoid ending up with the worst of both worlds! A detailed procedure to combine Fusion with OMT is provided. This really proves that Fusion is from a second generation because it builds on the existing methods.
(iv) Fusion incorporates checks for consistency and completeness because there is an underlying syntax and semantic foundation for all the notations. This makes it easy to create a supporting CASE tool.

(v) The main disadvantage is that the method requires a strong commitment to being systematic and rigorous, which may not always be appropriate.
(vi) Although Fusion is a method for sequential systems, it can be applied to the development of concurrent systems because many applications exhibit a very limited form of concurrency in which
-Concurrency is provided by an operating system.
-Processes are sequential programs.
-There is very limited use of dynamic process creation.
This kind of simple concurrent system can be viewed as a set of cooperating agents, each of which can be developed using Fusion.
The task of designing concurrent system architecture can not be achieved using Fusion, at least now, because it does not provide all the required notations and concepts such as synchronization.
(vii) One of the biggest limitations of Fusion is in the lifecycle model. This is not expressive enough to describe an arbitrary number of interleaved behaviors. In the book there is an example of a storage depot. If this model is slightly extended to have an arbitrary number of loading bays, the lifecycle model is not powerful enough to express the resulting behavior. Considering that a typical interactive user interface supports arbitrary number of windows at any one time, this seems a grave limitation.
(viii) Fusion does not really have a specification stage. The early lifecycle models in Fusion are supersets of real specification. You have to slip specification constraints in at the design stage. As an example, look at the object model. relationships in this model are bi-directional, this is fine for databases but it is not always necessary. Fusion requires you to make the decisions about whether these relationships really need to be bi-directional at the design stage. Really things like this should appear in a specification.

6 Methods evaluation

Throughout the previous example, we have been trying to show most of the features of each methodology in order to show the power of each; however, there are some features that were not shown due to the nature of the example. In this section, we will try to discuss the four methodologies in details in order to provide concrete guidelines on which is better and when. we will start the comparison by answering basic essential questions about each method, then we will proceed with the following issues: conceptual issues, general issues and finally the technique itself and we will provide tables of comparisons for each point of the discussion.

6.1 Essential questions

(i)Does the method make it easier to develop a system?
The method should provide a step-by-step guide for getting requirements to code. There should be checks for assessing when each step has been successfully completed. If multiple models are used then their interrelationship should be clear. It should be possible to get an overall view of the system and also detect inconsistencies.
OMT: The analysis stage provides a step-by-step process, and each step is supported by many heuristics. However, there are few rules for discovering inconsistencies between models.
Because there are no design models, design is essentially a process of coding the analysis models. The large gap between analysis and code can make this a daunting task.
Booch: The method comprises a set of notions together with a very ill-defined and loose process. There is a little help for the developer if he gets stuck.
RDD: As an exploratory techniques it is very powerful, but it bridges only part of the gap from requirements to code.
Fusion: The process is systematic and takes the developer from requirements through to code. The rigorous nature of the method means that it possible to detect inconsistencies between models.

(ii)Does the method make the system easier to maintain?

The method should allow the important decisions to be recorded. Thus the models should be intuitive, concise, and scaled up for large developments.

OMT: The object model can be used to record much structural information. The dynamic model is less successful because DFDs are essentially operational and are thus not good for specifying behavior. In practice, some OMT users omit the functional model. The models contain features for scaling up for larger developments.

Booch: Booch's notations are very comprehensive and can be used to document almost any aspect of a system. The main problem is that the notations tend to be verbose and their semantics are vague.

RDD: The index cards are the only record of the decisions taken and are clearly inadequate for maintenance purposes.

Fusion: The models can record all the key decisions made during development. However, it is still up to the developer to document the motivation and reasoning for the choices taken.

(iii)Does the method help produce software with good object-oriented structure?

The method should help the design of inheritance hierarchies and run-time structures. Two crucial problems are designing control flow and dynamic object creation.

OMT: Inheritance in analysis, design and implementation is fully treated by OMT. Object interaction is not supported by an explicit model; neither is dynamic object creation.

Booch: The Booch method contains a wealth of information on the semantics of object-orientation. The key mechanism concept explicitly addresses the flow of control issue.

RDD: The method helps with thd design of inheritance structures and is a powerful technique for designing object interaction.

Fusion: The method provides models specifically for designing control flow and dynamic object creation. However, it is the developer's responsibility to ensure that the decisions are well founded.

(vi)Does the method assist project management?

The method should have a systematic process with defined deliverables.

OMT: The deliverables from the analysis phase are well defined. However, because there are no design models, knowing when the analysis phase is complete and when design should begin can be a problem.

Booch: The Booch models constitute a well-defined set of deliverables. However, the lack of process limits the help that the method gives the project manager.

RDD: Only so far as the game playing helps team building!

Fusion: The systematic development of analysis and design models provides the basis for project management.

(v)Can the method be given effective tool support?

CASE tools help automate software development. Simple drawing tools require the notations to have a defined syntax. More powerful tools, such as those that allow requirements to be traced to code, depend on the semantics of the notations being defined.

OMT: OMT is supported by many CASE tools. The absence of a rigorous semantics limits them mainly to diagramming support.

Booch: Booch is supported by several diagramming tools, but it has the same problem of OMT.

RDD: Interestingly enough, it is not amenable to computer-based tools. One of the originators, Cunningham, originally developed the method using hypercards, but found index cards more portable and system independent.

Fusion: The syntax and semantics of the notations are fully defined. These are necessary for intelligent CASE support.

6.2 Conceptual issues

(i)*The degree Of object-oriented support*

Does the method support the usual meaning of the terms object (instance) and class ? Can a class be described in terms of its distinct parts: an interface and a body? Does the method support metaclasses as implemented in programming languages such as Smalltalk? Does the method support abstract classes. Refer to Table 1.

Table 1 Basic Object-Oriented elements

Method	Instance	Interface	Body	Abstract	MetaClass
Booch	Y	Y	Y	Y	Y
Rumbaugh	Y	Y	Y	Y	Y
Wirfs-Brock	Y	Y	Y	Y	N
Fusion	Y	Y	Y	Y	N

(ii)*What class to instance relationships does the method support?*

Does it support inheritance? If so, then does it support single or multiple inheritance? Dose the method distinguish between subclassing (that is implementation inheritance with restriction and redefinition as well as addition) and subtyping (with just addition)? Are associative relationships (with cardinalities) supported? What about aggregation relationships? Refer to Table 2.

Table 2 Object Relationships

Method	Single Inherit	Multiple	Subclassing	Association	Aggregation
Booch	Y	Y	N	Y	Y
Rumbaugh	Y	Y	N	Y	Y
Wirfs-Brock	Y	Y	N	Y	Partial
Fusion	Y	Y	N	Y	Y

(iii)*Does the method enable object lifetime to be charted?*

A method may be restricted to dealing with static systems of objects in which all objects have the same lifetime as the system. If this is not the case then the method must contain some facility for dynamically creating objects. Similarly for destroying an object. Can the method handle concurrently executed objects (active versus passive objects). Refer to Table 3.

Table 3 Object concurrency and lifetime.

Method	Active Instance	Passive	State	Persistence	Creation
Booch	Y	Y	Y	Y	Y
Rumbaugh	Y	Y	Y	Y	Y
Wirfs-Brock	N	Y	N	N	N
Fusion	N	Y	Y	Y	Y

(**iv**) *What kind of dynamic relations can exist between objects?*

Interactions between objects concern the kinds of message passing available. Message sending can involve either static or dynamic binding and can be polymorphic. Furthermore, does the method allow for representing only single messages or can sequences of associated messages be modeled ? Refer to Table 4.

Table 4 Message passing.

Method	Single Message	Multiple Message	Differentiation in Binding	Polymorphism
Booch	Y	Y	N	N
Rumbaugh	Y	Y	N	N
Wirfs-Brock	Y	Y	N	N
Fusion	Y	Y	N	Y

(**v**) *What models of communication does the method support?*

Synchronous communication requires the sender to support execution until the receiver accepts the message, whereas asynchronous communication allows the sender to continue. Does the notation distinguish between messages destined for local objects and messages destined for remote objects? Refer to Table 5.

Table 5 Communication semantics.

Method	Differentiation syn/asyn calls	Differentiation Local/remote
Booch	Y	Y
Rumbaugh	Y	N
Wirfs-Brock	N	N
Fusion	N	N

6.3 General method issues

(i)*Is the method oriented at real-time or information systems?*

Some methods support both, but most are heavily oriented towards one type of development or the other. How much of the software lifecycle does the method cover? Does it cover analysis and design? Refer to Table 6.

Table 6 Pragmatics

Method	Real time	IS	Analysis	Design
Booch	Y	-	N	Y
Rumbaugh	Y	Y	Y	Y
Wirfs-Brock	Y	Y	Y	Y
Fusion	Y	Y	Y	Y

(ii)*What are the typical problem domains where the method has been used?*

This is important because the method may be perfect but does not fit in a certain application. Refer to table 7.

Table 7 Problem domain

Method	Comments
Booch	MIS, telecommunications, manufacturing, defense, CASE tool development, petroleum, industry...
Rumbaugh	Compilers, graphics, user interfaces, databases, OO languages, simulation
Wirfs-Brock	All problem domains
Fusion	All problems except concurrent systems

(iii)*Are there guidelines for separating analysis from design?*

Does the methodology help with project management issues such as defining boundaries between analysis and design?

Booch and Rumbaugh differentiate between analysis and design; however, the boundaries are not well defined, while Wirfs-Brock consists mainly of two phases which are the exploratory phase (finding classes, their responsibilities and the collaborations between them) and the refinement phase (finding hierarchies and dividing the system into subsystems). Fusion has a unique feature that it does not bind the operations to classes during analysis, it does it during design so operations are not prematurely tied to classes.

(iv)*Does the method describe human factors concept?*

This includes any concept used in human factors modeling, which describes the people who operate within the problem domain, and their interface. Alternative names include: "OOD human interaction," "scenario" (painting), "user roles."

Refer to Table 8.

Table 8 Human factors concept

Method	Notation
Booch	N
Rumbaugh	Y, Scenarios
Wirfs-Brock	N
Fusion	Partial

(v)*What resources are available to support the method?*

Is the method fully supported by a CASE tool? Some methods are only partially supported. What hardware and software does the tool require? Does the environment support multi-user development? If there is CASE tool support, does it provide semantics processing such as simulation and code generation? What other resources are available to support the method such as published texts and so on?

(vi)*The ability to interwork with other CASE tools.*

This is important where a tool must integrate with other tools such as project management tools, or even a standard documentation system. Several standards are emerging in the area of CASE tool integration - such as the CDIF (CASE Data Interchange Standard).

Talking about Rational Rose again, it does not have such facilities.

(vii)*How scalable is the method?*

Scalability is concerned with whether techniques and notation can be used effectively on large systems. Notations need a mechanism for partitioning descriptions into smaller and more manageable modules and composing the whole from those modules. Some means of controlling the visibility of names across modules should also be provided.

This facility is available in Rose. It provides visibility of classes in different class categories.

(viii)*How well has the method been defined?*

Is there a well-defined set of steps by which the various techniques are strung together? Are objects traceable across the lifecycle? Does the method advocate only object-oriented techniques (pure) or does it suggest using some traditional structured techniques (hybrid)? Does the method support a defined notation? Refer to Table 9.

Table 9 Process.

Method	Well defined steps	Pure/ Hybrid	Notation	Traceable across life cycle
Booch	Partial	P	Y	-
Rumbaugh	Y	H	Y	Y
Wirfs-Brock	Y	P	Y	Y
Fusion	Y	H	Y	?

(ix)*Are there syntactic and semantic definitions for the notation, or do the syntax and semantics have to be deduced from examples?*
The syntax of a notation is a set of rules which describe the primitive components of a notation and the legal combination of those symbols. There are well-known techniques, such as the Backus-Naur Form, for formally defining textual syntax, but such techniques for diagrams are less established. There should, however, be a clear definition of the icons and their legal combinations. A defined syntax is a requirement for effective use and also for automated tool support. The semantics of a notation is a set of rules which gives the meanings of the syntactic primitives and their combinations. Fusion has a well-defined syntax and semantics which makes it unique among the other methods.

6.4 Techniques

(i)*Does the process address the architectural issues of the system under design?*
This is the capability of a method to split a system into subsystems. There are three facets in this:
-Decomposition into logically related parts.
-Collection of logical parts into modules which may be separately compiled.
-Deciding on the physical location where different parts belong.
Refer to Table 10.

Table 10 Architectural issues.

Method	Decomposition	Modularization	Distribution
Booch	Y	Y	Y
Rumbaugh	Y	Y	Y
Wirfs-Brock	Y	Y	Y
Fusion	Y	Y	N

(ii)*Does the method provide guidelines to help identify candidate objects?*
This is a most fundamental issue. You can't start manipulating and messaging objects until you've found some. An object can play different roles through associations with different objects. Does the method provide techniques for modeling the different views of an object. The methods discussed in this paper provide the facility for modeling different views of objects. OOD for example, has logical and physical views of the system each with a static and dynamic dimension.
(iii)*Does the method describe a process for generic object modeling? Does it cover all the activities?*
Refer to Table 11.

Table 11 Object identification activities

Method	Discover	Identify	Organize	Formalize	Review/ inspect	Refine	Agree
Booch	Y	Y	Y	N	Y	Y	Y
Rumbaugh	?	Y	Y	Y	Y	Y	N
Wirfs-Brock	not formally	not formally	not formally	not formally	not formally	not formally	not formally
Fusion	Y	Y	Y	Y	Y	Y	Y

(iv)*Does the method describe objects according to different classifications?*
Objects can be classified as a user interface object type, business logic object type(An object which provides the functionality of the application), information management object type(to provide persistent storage for the application), and finally legacy system object type(to encapsulate as existing non-object-oriented application or component, by defining an object-oriented interface so that it can be reused).
Refer to Table 12.

Table 12 Object types

Method	User interface	Business logic	Information management	Legacy system
Booch	Y	N	N	N
Rumbaugh	N	N	Y	Y
Wirfs-Brock	N	N	N	N
Fusion	N	N	Y	N

(v)*Which iteration strategy does the method advise between cycles and phases?*
The most commonly used iteration strategies are: once-only, evolutionary, rapid prototyping and incremental. The once-only iteration strategy assumes that the full production system can be built first time using one pass through all the development steps. The evolutionary iteration strategy assumes it will take several cycles to get a quality system, so it does not attempt to get it right first time. Rapid prototyping is a variant of evolutionary development in which the cycles are faster and the scope smaller than for evolutionary development. Finally, the incremental iteration strategy involves building the system in small increments. The following figure describes iteration strategies.
Refer to table 13.

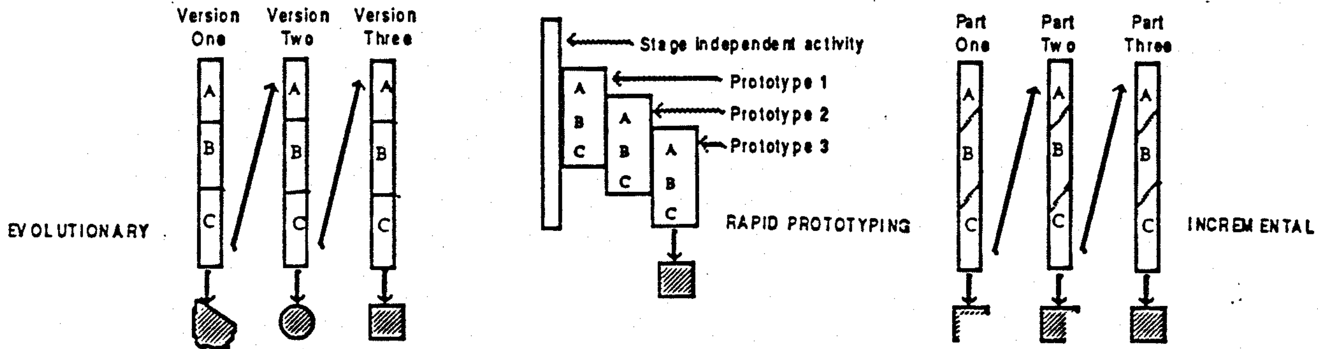


Table 13 Iteration strategy

Method	Once-only	Evolutionary	Rapid prototyping	Incremental
Booch	N	Y	Y	Y
Rumbaugh	Y	Y	Y	Y
Wirfs-Brock	N/A	N/A	N/A	N/A
Fusion	N	Y	Y	Y

(**vi**)*Does the method have development priority?*
Are their any concepts used to prioritize the order of development of the object types and functional capabilities identified in the analysis phase. Refer to Table 14.

Table 14 Development priority

Method	Comment
Booch	Y
Rumbaugh	Y, in system design
Wirfs-Brock	N
Fusion	?

(**vii**)*Does the method describe any new techniques for design modeling?*
Is there any unique feature in the design modeling?
Refer to Table 15.

Table 15 Unique design techniques

Method	Comment
Booch	N
Rumbaugh	Y
Wirfs-Brock	Y
Fusion	Y

(**viii**)*Which check pointing strategy (management control strategy between phases and cycles) does the method advise?*
The most commonly used check pointing strategies are: rubber-stamp, management reviews and risk driven. Rubber-stamp is a strategy in which management simple gives the go-ahead for the next stage or they abandon the project. Risk-driven is a strategy in which management consciously applies formal risk reduction techniques. Refer to Table 16.

Table 16 Check point strategies

Method	Rubber-stamp	Management review	Risk driven
Booch	N	Y	Y
Rumbaugh	not addressed	not addressed	not addressed
Wirfs-Brock	indeterminate	indeterminate	indeterminate
Fusion	N	Y	Y

(ix)*What kind of rule concepts does the method use and support?*
A rule is a policy or condition that must be satisfied. Rules can be constraints or assertions(precondition and postcondition and invariant). Refer to Table 17.

Table 17 Rules concept

Method	Constraint	Assertion
Booch	Y	N
Rumbaugh	Y	Y
Wirfs-Brock	N	N
Fusion	Y	Y

(x)*Does the method include any schema concepts?*
A schema is a collection of object types and other schemas which constitute some form of operational system. Every schema identifies a list of object types, some schemas impose structure on this list. Refer to Table 18.

Table 18 Schema support

Method	Comment
Booch	N
Rumbaugh	Y
Wirfs-Brock	Y
Fusion	Y

(xi)*Are there any guidelines for developing inheritance hierarchy?*
Associated with this is the issue of designing for reuse. Reusable components and designs must be developed. A method needs to provide explicitly activities which are intended to identify reusability and support the development of reusable components and design. Refer to Table 19.

Table 19 Techniques.

Method	Class identification	General/ Special	For reuse	With reuse
Booch	Y	Y	-	-
Rumbaugh	Y	Y	Partial	N
Wirfs-Brock	Y	Y	Y	-
Fusion	Y	Y	Y	Y

(**xii**)*What guidelines does the method provide to ensure quality?*
Are there guidelines on consistency, completeness, coupling and cohesion, and design extensibility. Refer to table 20.

Table 20 Quality issues.

Method	Completeness	Consistency	Coupling/ cohesion	Extensibility
Booch	N	N	Y	N
Rumbaugh	Y	Y	Y	Y
Wirfs-Brock	N	Y	Y	Y
Fusion	Y	Y	N	Y

7 Practical solution

As we have seen each methodology has its own advantages and disadvantages. Most of methods have solved a piece of the puzzle. The only way to get a robust complete method is to mix and match the best from each. It helps a lot when you can find a method that offers a good starting point. we think that Fusion, although it has some problems, can be considered a good starting point and the following additions, from other existing methods, can help to overcome the limitations. Also, some modifications can improve the method.

7.1 Suggested components of a method

After describing and using the four methods, OMT, OOD, RDD and Fusion, it is clear that there are some guidelines and similarities between the methods, especially between OOD, OMT and Fusion. The basic components that a method should include are:

(**i**)A way to describe the classes, their states and the relationships among them.

(**ii**)A systematic way to capture the dynamic behavior of the system (interactions among objects and states of the object).

(**iii**)A notation to describe objects visibility.

(**iv**)Sufficient set of notations to support most kinds of systems (concurrency and synchronization for example).

(**v**)A notation to describe the allocation of processes to processors.

7.2 Putting things together

So let us try to get the best of each methodology to provide the basic things we need:

(i)Classes and their relationships

CRC cards, in our opinion, are the best out of the four in describing the classes. They are simple and extremely useful especially if Smalltalk is used for implementation; however, they lack a way to model the relationships between classes. So we suggest the combination of CRC cards and object model of Fusion . As for the state transition diagrams for each class, we suggest OOD's approach for its simplicity (figures 10,11), because using regular expressions, the approach in Fusion(figure 18), is not practical, they are harder to understand and to implement.

(ii)Dynamic behavior

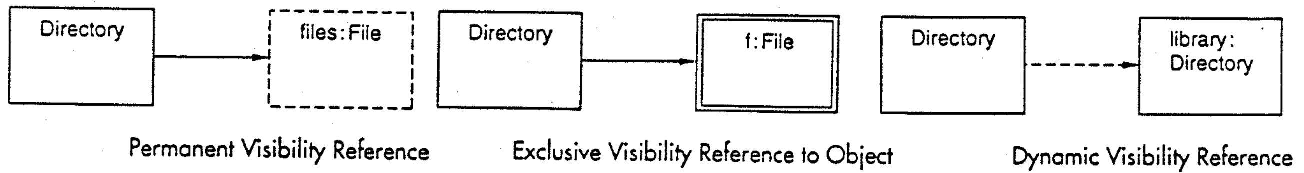
OMT's event trace diagrams, OOD's object timing diagrams and Fusion's object interaction graphs can be seen as three equivalent ways of describing the dynamic behavior. We prefer event trace diagrams (figure 2) since they are easier to draw and more understandable and readable than object timing diagrams. The language of Fusion's object interaction graphs is not quite strong enough. For even better tool for object interactions, Jacobson's interaction diagrams should be used [4].

(iii)Process diagrams

OOD is unique in describing the system in terms of physical and logical views. We suggest that OOD's process diagram is used here to describe the allocation of processes to processors and devices (figure 13).

(iv)Visibility graphs

Fusion is unique in describing objects visibility, this feature was ignored by most of other methods except Booch. The following figures show simple examples of visibility graphs.



(v)Things to be added

Still there are other needed features to be added or to replace the existing ones in our choice in order to cover the whole lifecycle.

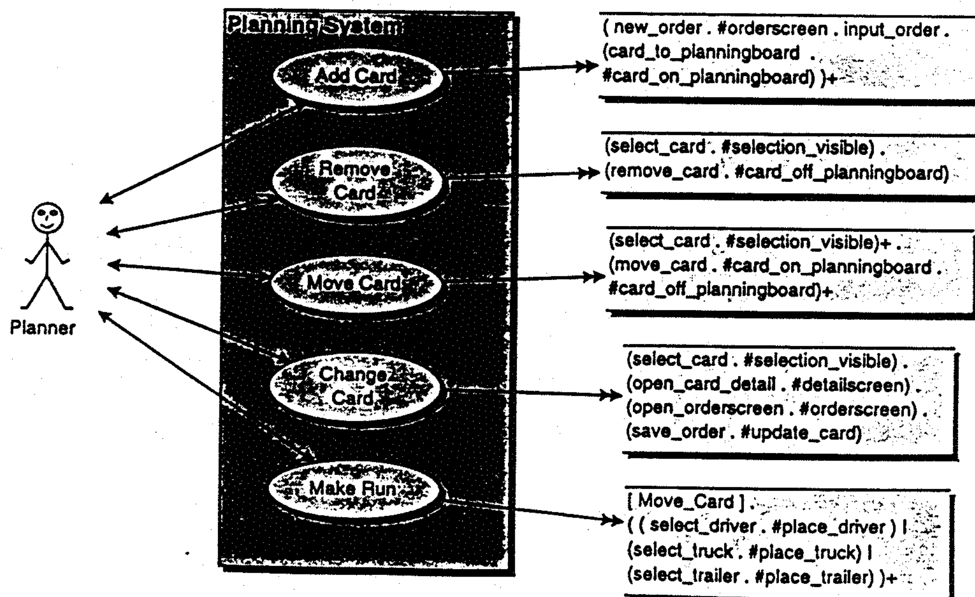
Features to be added:

-Additions to interaction graphs to model protocols and distributed objects.

-Additions to visibility graphs to model protocols and distributed objects.

-Additions to class descriptions to notate protocols, distributed objects and interactions to GUI objects.

-User interaction graphs which is a combination of use cases, lifecycle model and prototypes that deal with user interaction (to catch the system operations. e.g. coming from buttons, menu item, etc.). the following figure is an example of user interaction graphs. This kind of graphs was presented in NEXTSTEP EXPO'94 and is currently used by a company called SHARED OBJECTIVES in the Netherlands.



8 Conclusions

None of the major object-oriented methods are perfect, some tailoring may be needed. So the best way is to start with a methodology and keep on adding features to it according to your requirements and environment.

Good CASE tools are certainly needed to support the methodologies and provide automatic consistency checks and smooth transitions from phase to phase.

"Science is always wrong-it never solves a problem without creating ten more!."* This statement summarizes the current situation of object-oriented methodologies. One of the main advantages of them is that they provide a clearer and more natural means of communication between analysts and users; however, they created some confusion since almost each one uses different notations, and more methods and notations continue to appear in the market which does not solve the problem, it even makes it worse! At this stage in the life of Object Technology, it is in the best interest of the software development community for notations to be standardized across the various methodologies.

Fusion proves that we do not need to create new object-oriented methods from scratch because it uses the existing notations, with slight modifications, successfully. So it is better now to try to combine the best of the methods in a coherent way than to waste time building new components. Even if Fusion does not come with all the solutions, at least it is a good step toward standardization of object-oriented notations, so we can say that it paves the way for more productive second-generation methods to come. So we can say that we are on the right track of evolution.

More emphasis should be given for maintenance and reuse. Clearly, distributed object-oriented systems require extensions to existing methods. This is where the third generation of methods should start.

References

[1] Booch, G. Object-Oriented Analysis and Design with Applications. Benjamin/Cummings, 1994.

[2] Booch, G., Fowler, M., Goldberg, A., and Rubin, K. "Object-Oriented Analysis and Design, Finding your Path." SIGS Publications 1993.

[3] Coleman, D. Object-Oriented Development. The Fusion Method. Prentice Hall,1994

[4] Fowler, M. "A Comparison of Object-Oriented Analysis and Design Methods." Object World July 1992.

[5] Hutt, Andrew. Object Analysis and Design. Comparison of Methods. A Wiley-QED, 1994.

[6] Jacobson, I., Christerson, M., Jansson, P., and Overgaard, G. Object-Oriented Software Engineering. A Use Case Approach. Addison-Wesley,1992.

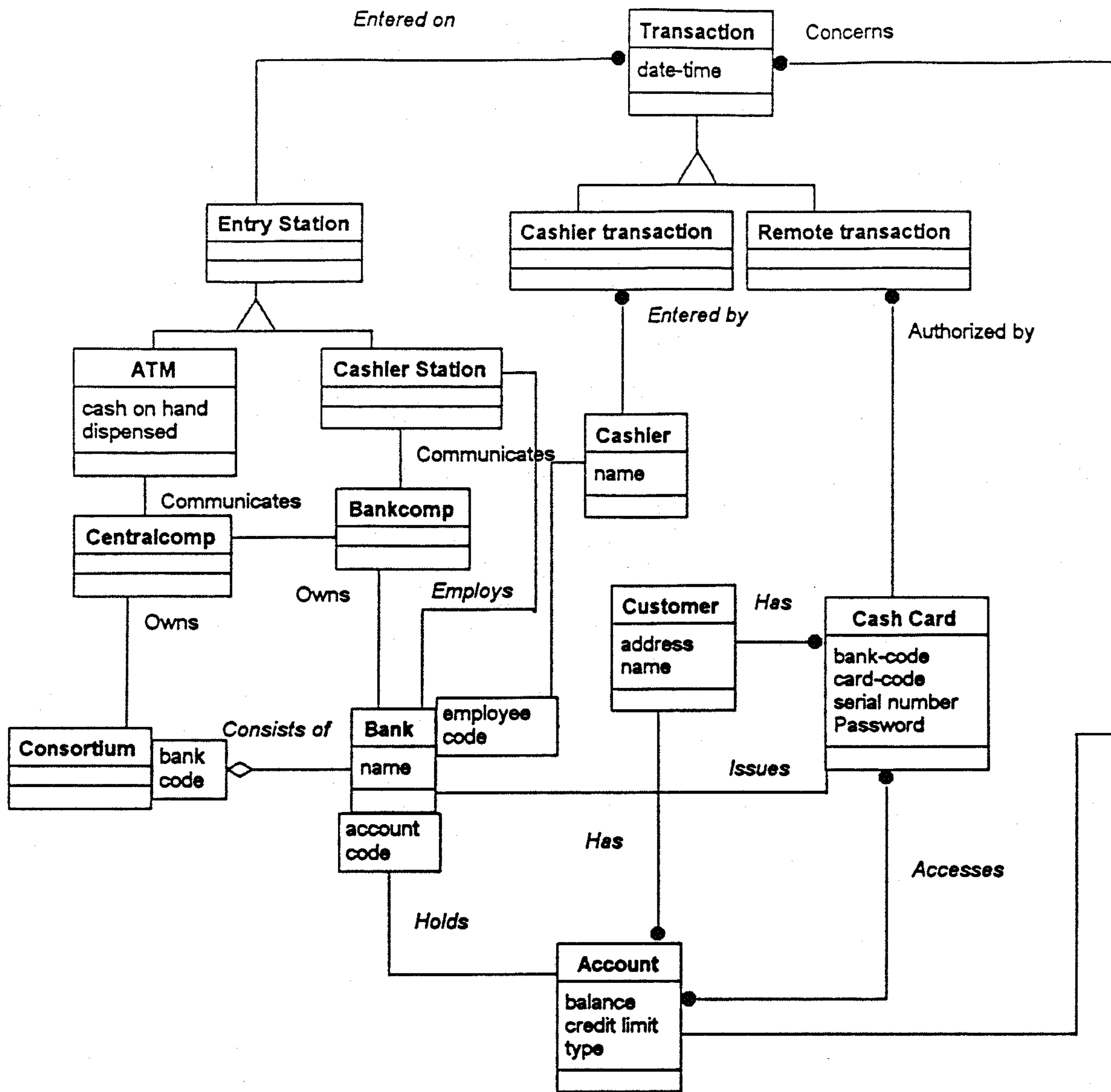
[7] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. Object-Oriented Modeling ad Design. Prentice Hall, 1991.

[8] Wirfs-Brock, R., Wilkerson, B., and Wieneer, L. Designing Object-Oriented Software. Prentice Hall, 1990.

* George Bernard Shaw

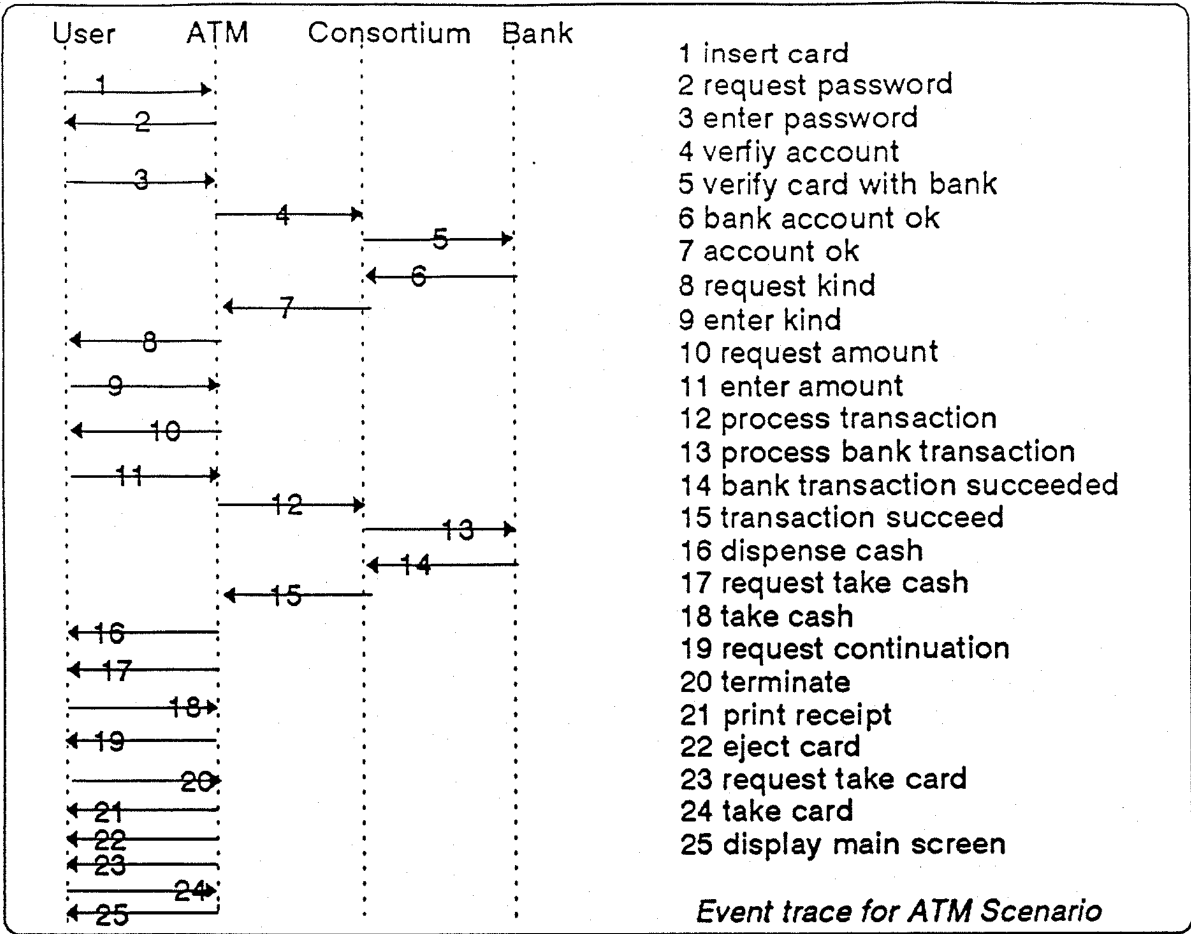
Appendix A

This appendix includes the figures of the four solutions



Object Model Diagram

Figure 1



Event trace for ATM Scenario

Figure 2

Module: atm Sheet: 2: Dynamic Model Panel: (0,0) Time: Sat Mar 26 20:16:57 1994

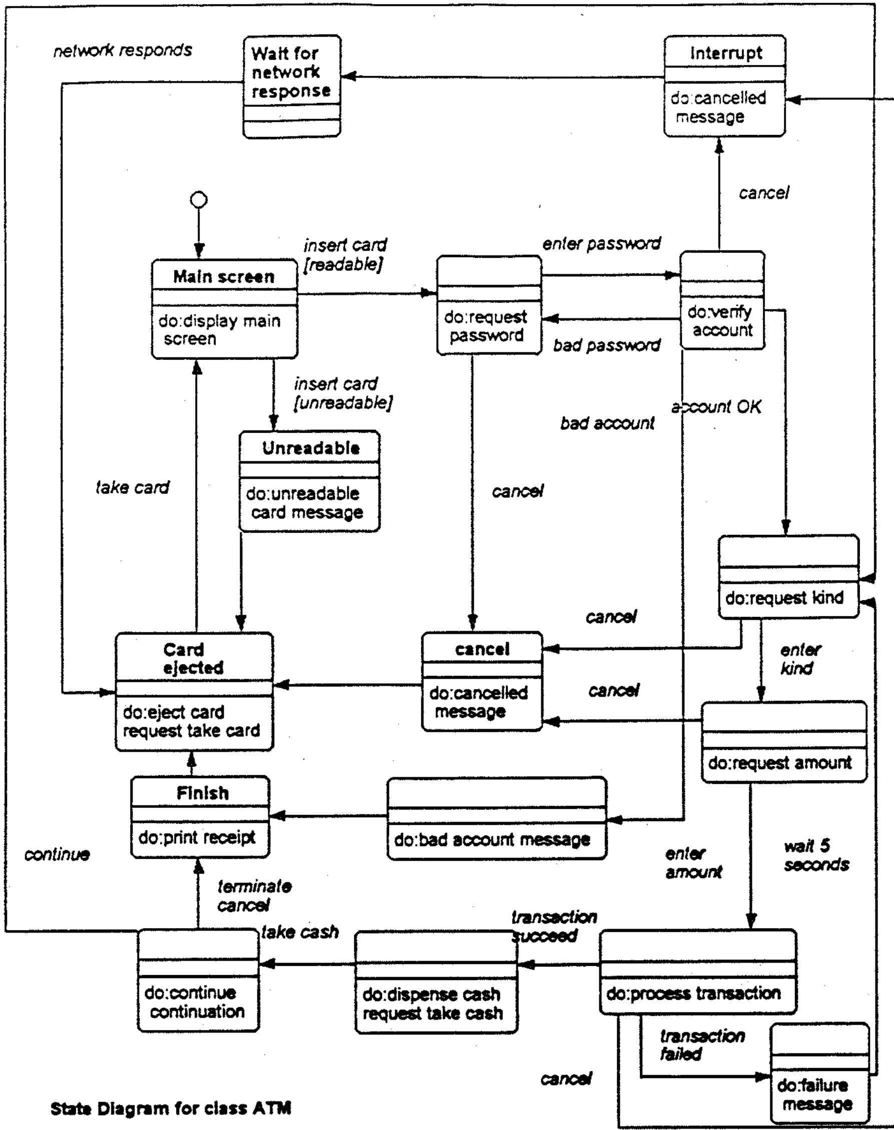
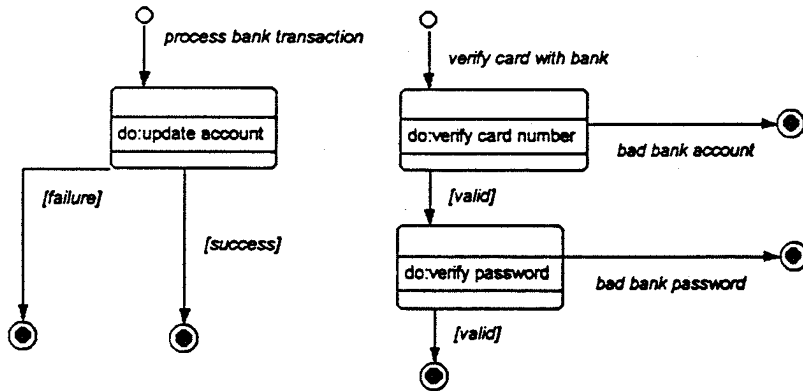
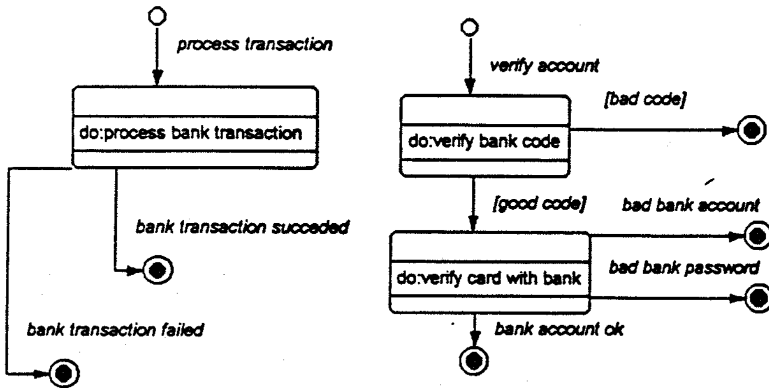


Figure 3



State Diagram for Class Bank

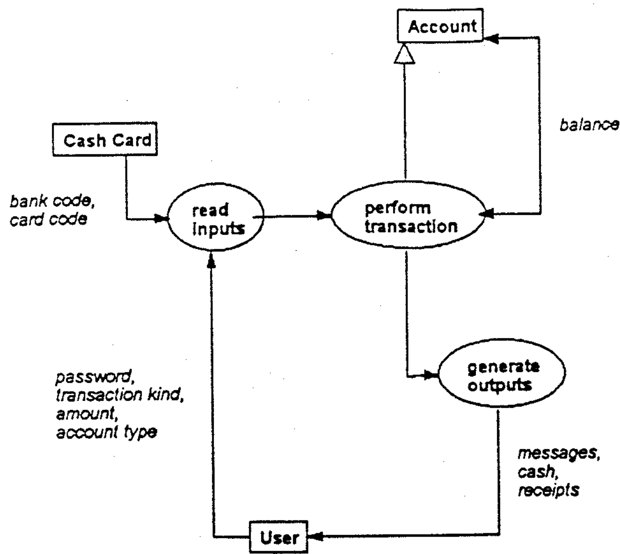
Figure 4



State Diagram for class Consortium

Figure 5

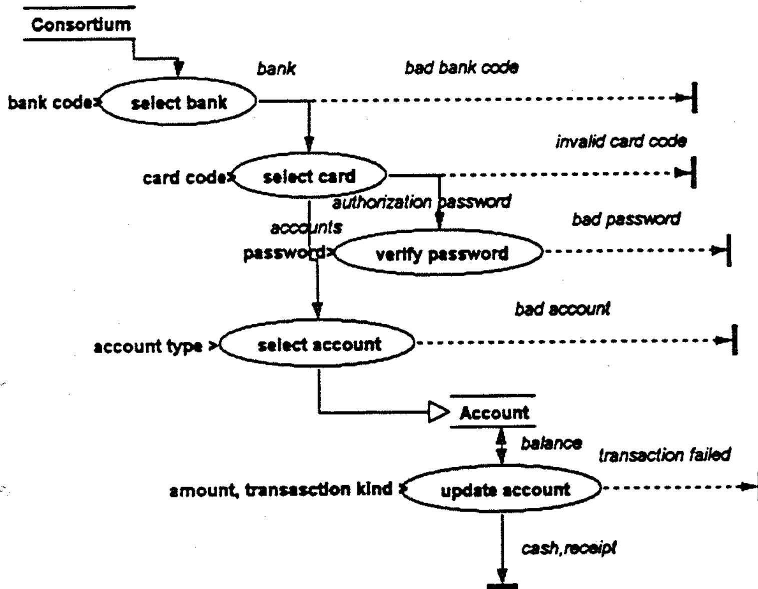
Module: atm Sheet: 3: Functional Mode Panel: (0,0) Time: Sat Mar 26 21:30:07 199



Top level data flow diagram for the ATM

Figure 6

Module: atm Sheet: 4: dfd2 Panel: (0,0) Time: Sat Mar 26 22:14:42 1994



Data flow diagram for ATM perform transaction process

Figure 7

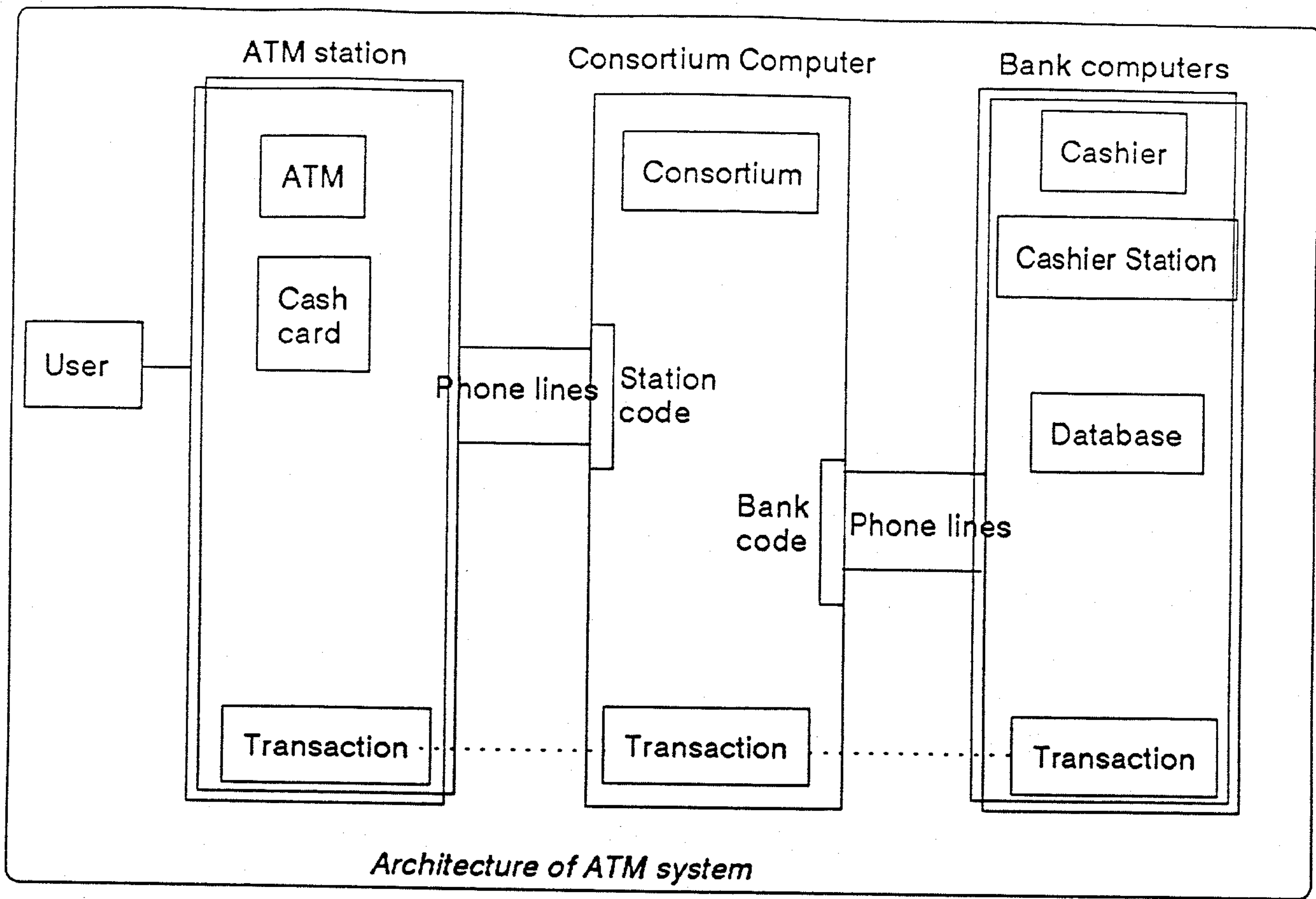
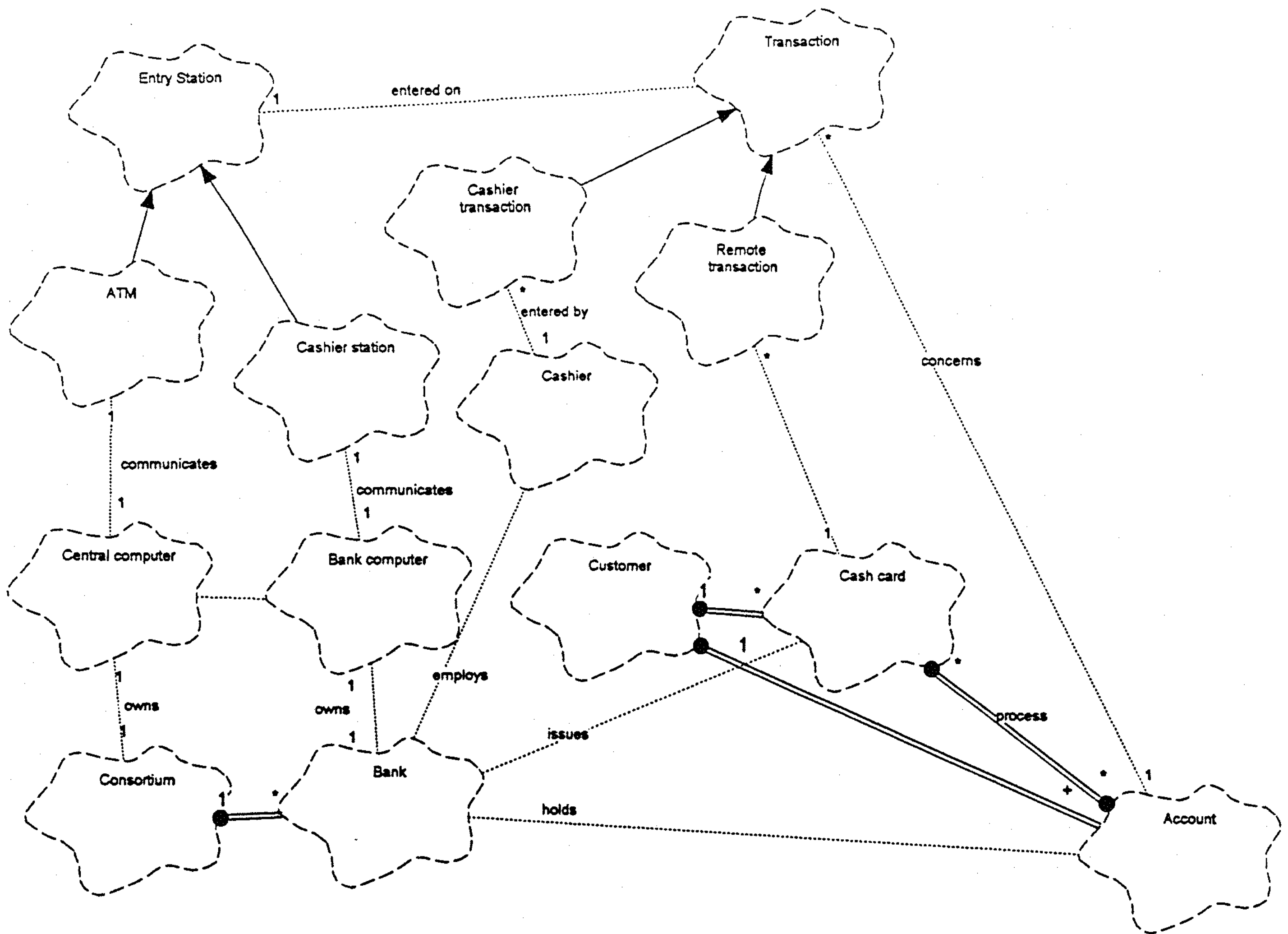
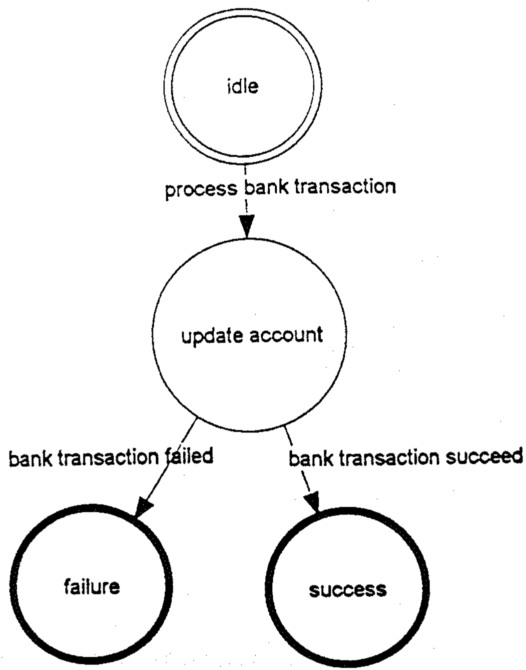


Figure 8



Class diagram for ATM



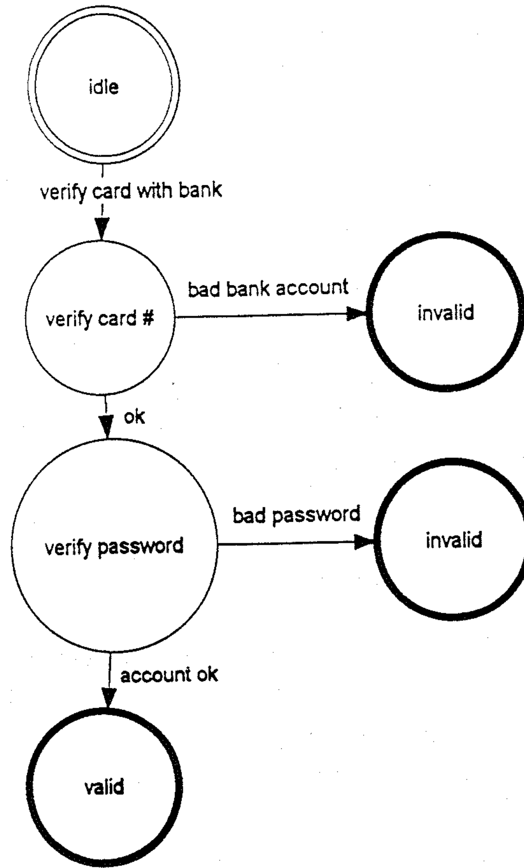
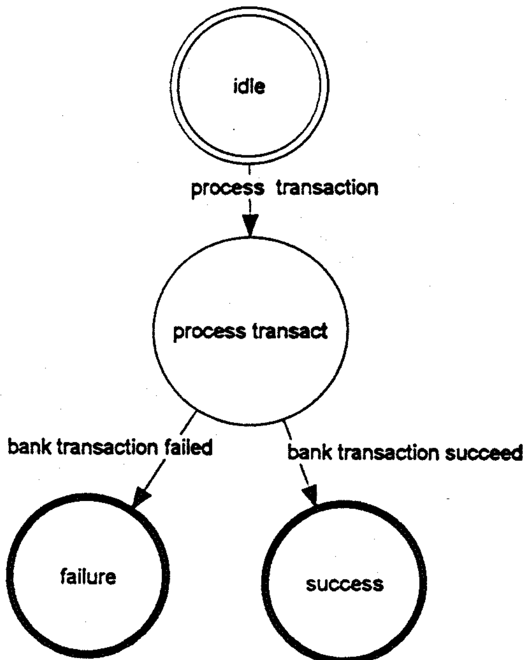


Figure 10

State transition diagram for class bank



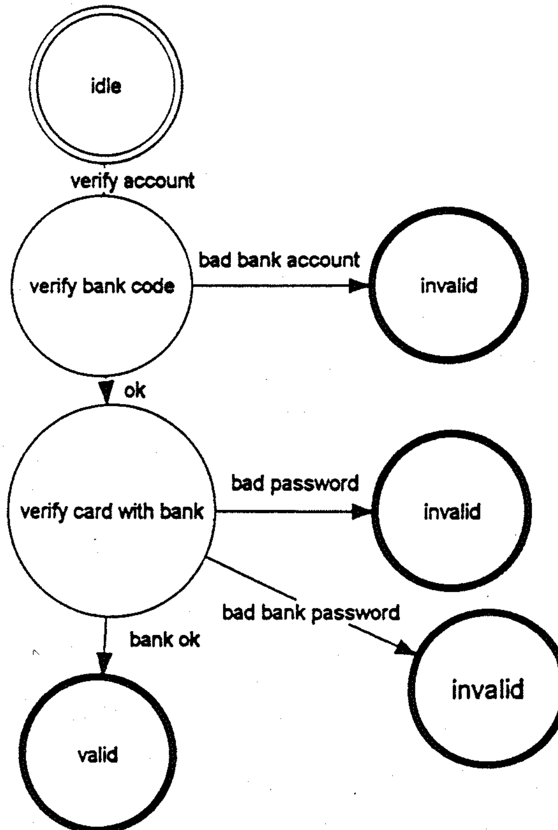
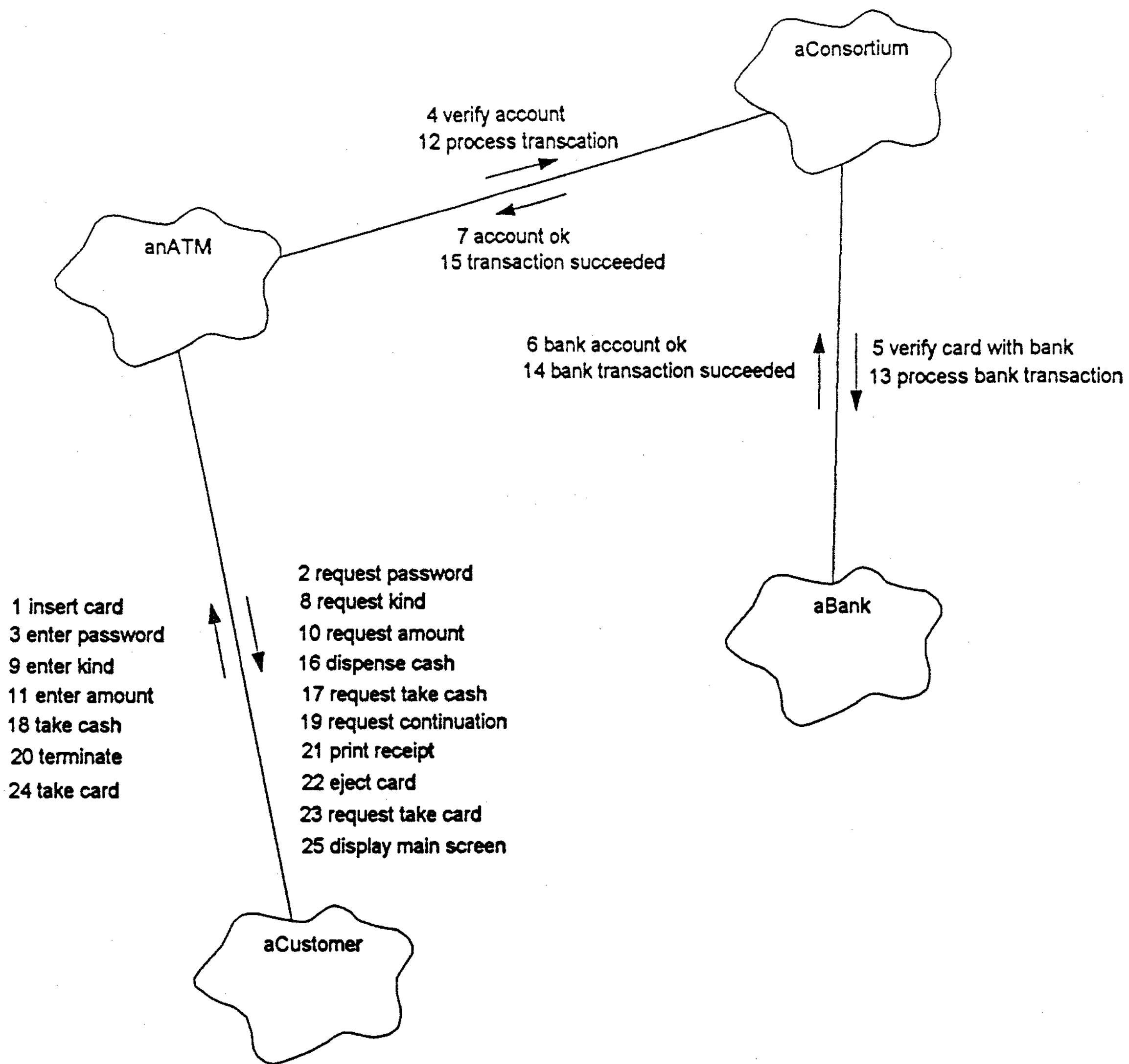


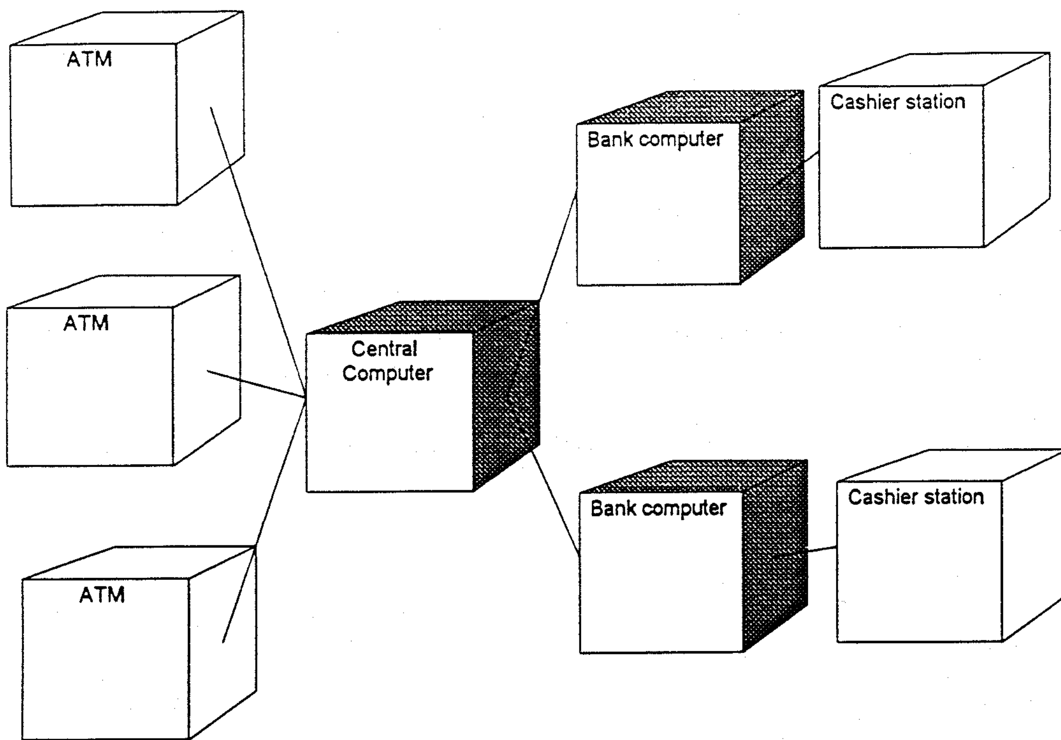
Figure 11

State transition diagram for class consortium



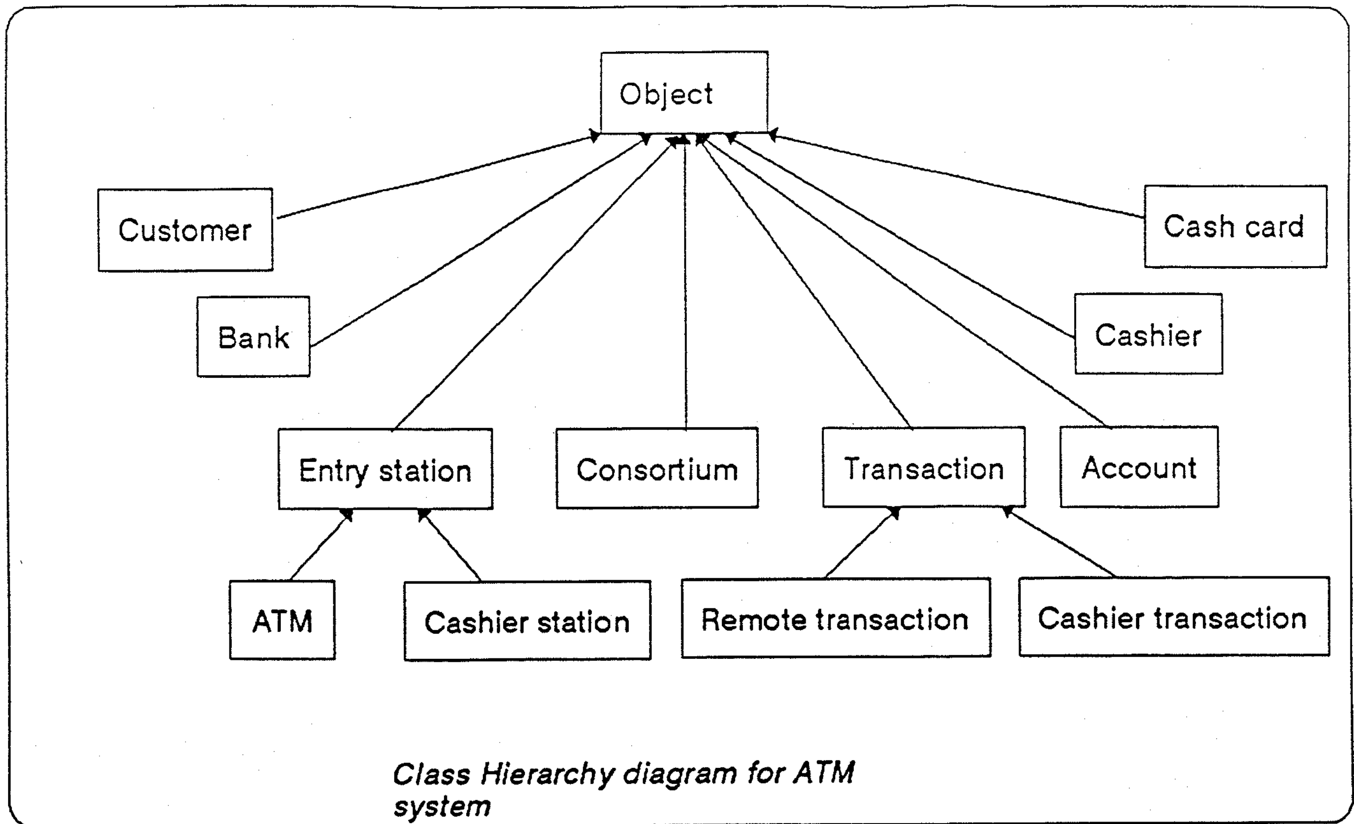
Object timing diagram

Figure 12



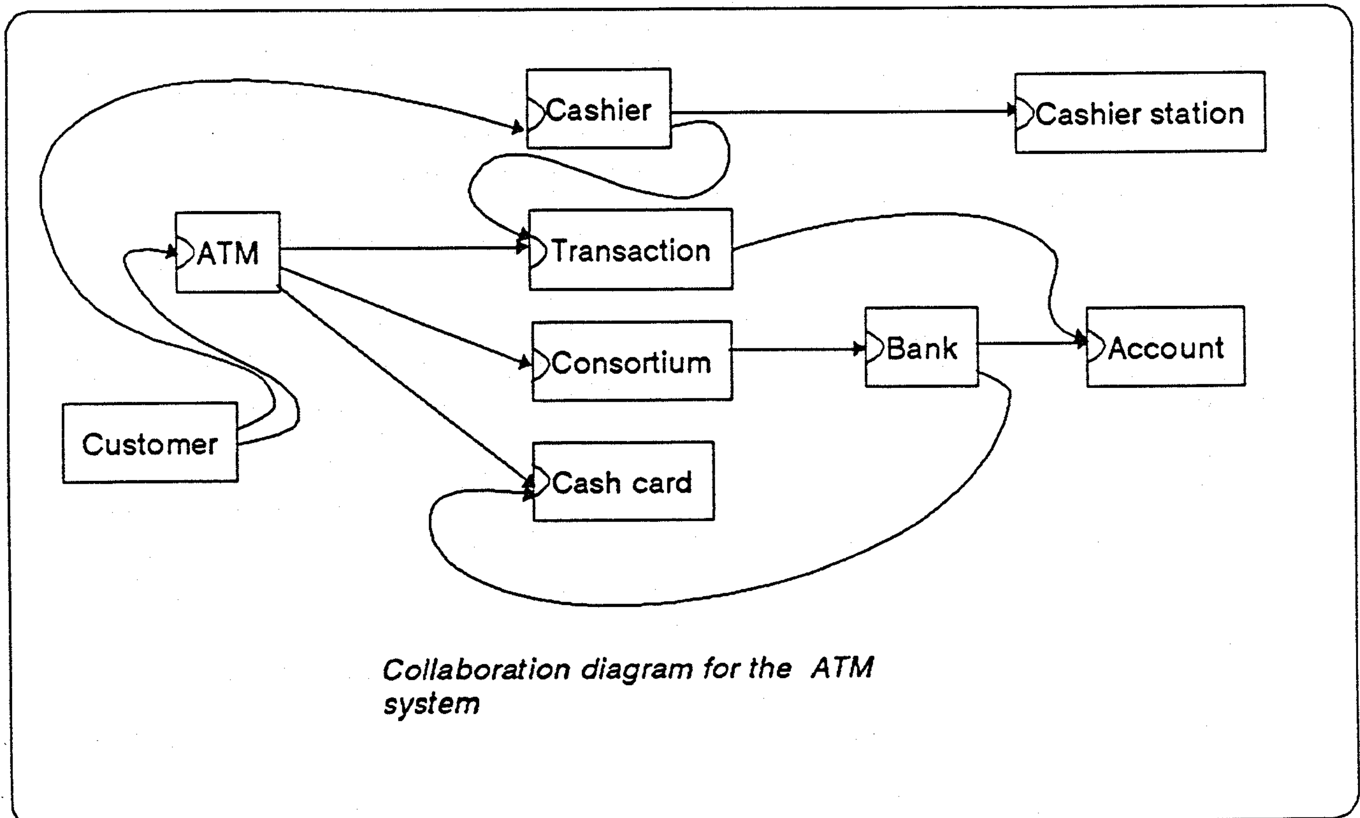
Process diagram for ATM

Figure 13



Class Hierarchy diagram for ATM system

Figure 14



Collaboration diagram for the ATM system

Figure 15

Class : Account

Responsibilities	Collaborators
-Keeps track of its balance. -Keeps rack of its credit limit. -Keeps track of its type. -Accepts deposits. -Accepts withdrawals.	-None.

Class : ATM

Responsibilities	Collaborators
-Accepts cash cards. -Gets information from customers. -Prints receipts.	
-Creates and initiates transactions.	-Transaction.
-Sends transactions to consortium for validation and processing. -Prints receipt.	-Consortium.
-Ejects cash card.	-Cash card.

Class :Bank

Responsibilities	Collaborators
-Keeps track of its name.	
-Creates and initiates accounts. -Keeps track of its employees.	-Account.
-Issues cash cards.	-Cash card.

Class :Cash card

Responsibilities	Collaborators
-Keeps track of its password. -Keeps track of its bank code, serial number. -Keeps track of its cash limits.	-None.

Class :Cashier

Responsibilities	Collaborators
-Keeps track of his name.	
-Enters transactions and dispenses cash.	-Transaction.
-Prepares receipts.	-Cashier station.

Class :Cashier station

Responsibilities	Collaborators
-Prints receipts.	-None.

Class : Consortium

Responsibilities	Collaborators
-Operates the ATM network.	
-Keeps track of its banks.	-Bank.

Class : Customer

Responsibilities	Collaborators
-Keeps track of his name, personal information..	
-Uses ATM to cash money.	-ATM.
-Contacts cashiers for service.	-Cashier.

Class : Transaction

Responsibilities	Collaborators
-Keeps track of its kind, date, time. and amount.	
-Operates on corresponding accounts	-Account.

Class : Cashier transaction

Responsibilities	Collaborators
- Keeps track of transactions entered by cashiers.	- Account.

Figure 16

Operation: withdraw_cash

Description: Requests an amount of cash to be taken from a given account. Cash is dispensed only if account has sufficient funds.

Reads: supplied acc_number, supplied request.

Changes: account with account.number equal to acc_number.

Sends: customer: {insufficient funds}, cash_dispenser: {dispense_cash}

Assumes: acc_number is valid. The account is not overdrawn.

Result: If (initial account.balance >= request) then
account.balance has been reduced by request, and
dispense_cash amount has been sent to cash dispenser.
Otherwise, insufficient_funds has been sent to customer.
The account is not withdrawn.

Schema for withdraw_cash
Operation Model

Figure 17

Interface Model Notations
Life-cycle model

Life-cycle [name :] Regular-Expression

(LocalName = Regular_Expression)*

Regular_Expressions:	Name	Any event name, local name, or output event.
	Concatenation	x.y
	Alternation	x|y
	Repetition	
	0 or more	x^*
	1 or more	x^+
	Optional	[x]
	Interleaving	||
	Grouping	(x)

Lifecycle: Banking_system: Initialization. ((Transaction | Enquiry)* || PrivEnquiry*)

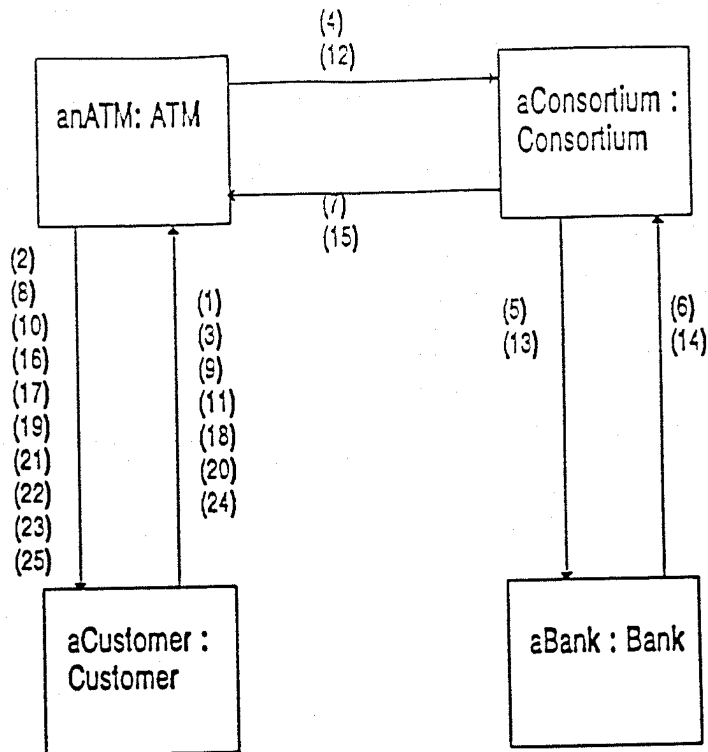
Initialization = open_account.#account_number . Enquiry* . deposit_money

Transaction = withdraw_cash . (#dispense_cash | #insufficient_funds) | deposit_money

Enquiry = check_balance . #current_balance

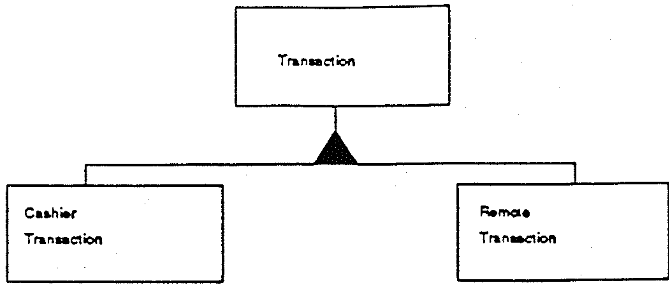
PrivEnquiry = authorize . (#confirm | #deny) . (inquire . #amount)* . finish

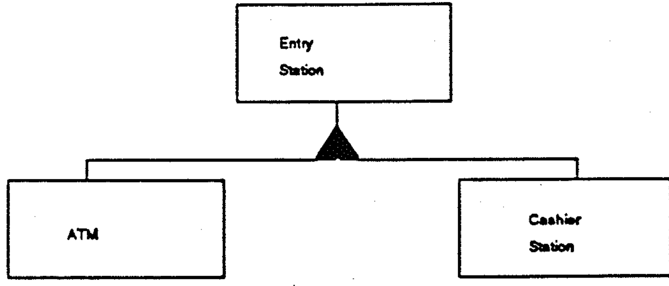
Life cycle Model



Object Interaction Graph Description

Insert card	(1)
Request password	(2)
Enter password	(3)
Verify account	(4)
Verify card with bank	(5)
Bank account OK	(6)
Account OK	(7)
Request kind	(8)
Enter kind	(9)
Request amount	(10)
Enter amount	(11)
Process transaction	(12)
Process bank transaction	(13)
Bank transaction succeeded	(14)
Transaction succeeded	(15)
Dispense cash	(16)
Request take cash	(17)
Take cash	(18)
Request continuation	(19)
Terminate	(20)
Print receipt	(21)
Eject card	(22)
Request take card	(23)
Take card	(24)
Display main screen	(25)





Inheritance Graphs

Figure 20

TR-234 **String Taxonomy Using Learning Automata**
B. John Oommen and Edward V. de St. Croix, March 1994

TR-235 **Distributed Cyclic Reference Counting**
Frank Dehne and Rafael D. Lins, March 1994

TR-236 **Exact and Approximate Computational Geometry Solutions of an Unrestricted Point Set Stereo Matching Problem**
Frank Dehne and Katia Guimaraes, March 1994

TR-237 **Scalable and Architecture Independent Parallel Geometric Algorithms with High Probability Optimal Time**
Frank Dehne, Claire Kenyon and Andreas Fabri, March 1994

TR-238 **Finding the Extrema of a Distributed Multiset**
Paola Alimonti, Paola Flocchini and Nicola Santoro, March 1994

TR-239 **Killing Two Birds with One Stone**
Evangelos Kranakis, Danny Krizanc, Anil Maheshwari, Jörg-Rüdiger Sack, Jorge Urrutia, April 1994

TR-240 **Some Computational Problems on Central Simple Algebras over Q**
Vincenzo Acciaro, April 1994 (Not available)

TR-241 **Extending Cryptographic Logics of Belief to Key Agreement Protocols**
Paul C. van Oorschot, May 1994

TR-242 **Modern Key Agreement Techniques**
Rainer A. Rueppel and Paul C. van Oorschot, May 1994

TR-243 **On Unifying Some Cryptographic Protocol Logics**
Paul F. Syverson and Paul C. van Oorschot, May 1994

TR-244 **Efficient DES Key Search**
Michael J. Wiener, May 1994

TR-245 **Optimal Fault-Tolerant Leader Election in Chordal Rings**
Bernard Mans and Nicola Santoro, May 1994

TR-246 **Information Privacy in Canada (Legislation in the Face of Changing Technologies)**
Václav Matyás, Jr., June 1994

TR-247 **Randomized Encryption with Insertions and Deletions Based on Probability Distributions**
B. John Oommen and P.C. van Oorschot, June 1994

TR-248 **On the Number of Directions in Visibility Representations of Graphs**
Evangelos Kranakis, Danny Krizanc and Jorge Urrutia, July 1994

TR-249 **Parallel Collision Search with Application to Hash Functions and Discrete Logarithms**
Paul C. van Oorschot and Michael J. Wiener, July 1994

TR-250 **Graph Partitioning Using Learning Automata**
B. John Oommen and Edward V. de St. Croix, July 1994

TR-251 **Cellular Automata in Fuzzy Backgrounds**
G. Cattaneo, P. Flocchini, G. Mauri, N. Santoro, C. Quaranta Vogliotti, August 1994

TR-252 **CATS 94**
September 1994

TR-253 **Bubbles: Adaptive Routing Scheme for High-Speed Dynamic Networks**
Shlomi Dolev, Evangelos Kranakis and Danny Krizanc, September 1994

TR-254 **Object-Oriented Methodologies: A Matter of Perception**
John Pugh and Amir Zeid, October 1994