ISSN(Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 4, Issue 2, February 2015**

# A Novel Dynamic Object Oriented Model for Automated Credit Card Management System

R.S.Venkatesh[1], K.Deepika[2], R.Poornima[3], Prof.S.Balamurugan[4], Prof.M.Sowmiya[5]

Department of IT, Kalaignar Karunanidhi Institute of Technology, Coimbatore, TamilNadu, India[1,2,3,4,5]

**ABSTRACT**: The paper aims to employ a modeling language aimed to provide a unified framework for representing online system personal finance management system. The proposed modeling methodology is based on the principle of object orientation [3], which allows describing both software and functionalities. Furthermore, it is illustrated how the well-known object-oriented specification language unified modeling language can be adopted, to provided an adequate formalization of its semantics, to describe structural and behavioral aspects of online personal finance management system related to both logical and physical parts. It is needed to implement the software on the basis of Object Oriented Model developed. Flaw in modeling process can substantially contribute to the development cost and time. The operational efficiency may be affected as well. Therefore special attention should be paid to the correctness of the models that are used at all planning levels and hence Unified Modeling Language (UML) [4] plays an important role.

**KEYWORDS**: Unified Modeling Language, Object Oriented Model, Online Personal Finance Management System.

## I. INTRODUCTION

Most of the effort to date in object oriented community has been focused on programming language issues. Object-oriented programming languages are useful in removing restrictions due to the inflexibility of traditional programming languages. In a sense, however, this emphasis is a step backwards for software engineering by focusing excessively on implementation mechanisms, rather than the underlying thought process that they support.

The real pay-off comes from addressing front-end conceptual issues, rather than back-end implementation issues. Design flaws that surface during implementation are most costly to fix than those that are found earlier. Focusing on implementation issues too early restricts the design choices and often leads to inferior product. An object-oriented development approach encourages software developers to work and think in terms of application domain through most of the software engineering lifecycle.

Object-oriented development [5] is a conceptual process independent of programming language until final stages. Object-oriented programming is fundamentally a new way of thinking and not a programming technique. Its greatest benefits come from helping specifiers, developers, and customers express abstracts concepts clearly and communicate them to each other. It can serve as a medium for specification, analysis, documentation and interfacing as well as programming.

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. UML includes a set of graphical notation techniques to create abstract models of specific systems, referred to as UML model. The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The Unified Modeling Language offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components. UML is officially defined by the Object Management Group (OMG) [3] as the UML metamodel, a Meta-Object Facility metamodel (MOF). Like other MOF-based specifications, UML has allowed software developers to concentrate more on design and architecture.

UML models may be automatically transformed to other representations (e.g. Java) by means of QVT-like transformation languages, supported by the OMG. UML is extensible, offering the following mechanisms for

ISSN(Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2015

customization: profiles and stereotype. The semantics of extension by profiles have been improved with the UML 1.0 major revision. It is very important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphical representation of a system's model. The model also contains a "semantic backplane" — documentation such as written use cases that drive the model elements and diagrams. UML diagrams represent three different views of a system model:

Functional requirements view: Emphasizes the functional requirements of the system from the user's point of view. And includes use case diagrams.
Static structural view: Emphasizes the static structure of the system using objects, attributes, operations and relationships. And includes class diagrams and composite structure diagrams.
Dynamic behavior view: Emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. And includes sequence diagrams, activity diagrams and state machine diagrams.

## II. CLASS DIAGRAM

UML class diagrams [3] are the mainstay of object-oriented analysis and design. UML class diagrams show the classes of the system, their interrelationships (including inheritance, aggregation, and association), [7] [8] [9] [21] and the operations and attributes of the classes. Class diagrams are used for a wide variety of purposes, including both conceptual/domain modeling and detailed design modeling.
Customer banks with the Financial Institution through its banking site via network. Both of them use computer for communication. Credit cards are issued to the customers via the credit card company, and they in turn aggregate to form credit unions. A customer can own many credit cards of different companies.

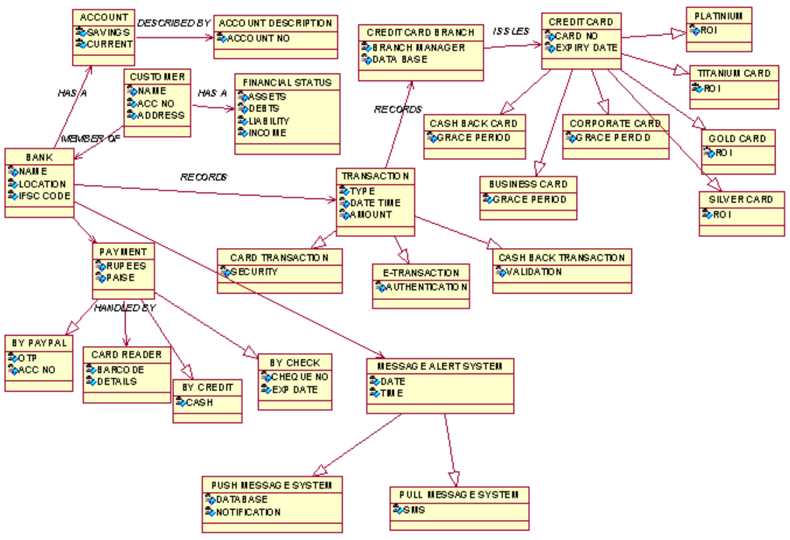

Fig 1. Class Diagram for Credit Card Management System

Financial Institution provides customer the facility to maintain an account. A customer can possess multiple accounts in various institutions. These accounts aggregate to form account aggregation, which is yet another facility provided by the financial institution. Students account and Retirement account are a kind of account (inherited from the account class). The customer can maintain these accounts if they possess proper eligibility. The customer can transfer funds to a creditor he wishes after the validation of the account numbers. The customer can perform transaction if he has a valid account in the institution. He can also perform foreign exchange transaction which is the inherited form of transaction. He can check the status of the transaction which he/she has performed (scheduled, processed or pending). From this status the customer can know about the completion of transaction. The customer is also provided with the facility of viewing the account history. The customer can make payments which can be of any type (multiple, recurring, email,

ISSN(Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 4, Issue 2, February 2015**

electronic or cheque according to his wish. The customer can also pay bills (e-bills/ overdue bills) online. The financial institution issues alerts incase of overdue bills. The alert can be web/email alert. The financial institution issues receipts for the confirmation of bill payments to the email id of the customer. If the customer does not prefer online payments, the financial institution issues a cheque. The customer can perform mortgage online through the brokerages. The can participate in auctions. The financial institution provides loan facilities after the validity of information provided. The loans include home loan, equity loan, employee benefit loan and educational loan. The financial planning calculators, which is an exquisite facility provided by the financial institution provides the growth of the interest rate and the amount that the customer has in his stock. The financial institution provides customer service for easy rectification of the problems faces by the customers, if any. It provides security to the customers by periodically changing the user id and password on customer's request. It also assures privacy against harassment and ensures privatization of the customer information. This is depicted in Fig. 1 (at Appendix)

## III. USE CASE DIAGRAM

Use case diagrams [16] [19] [20] model the functionality of a system using actors and use cases. Use cases are services or functions provided by the system to its users. The components in a use case diagram include:

- Actors: Actors represent external entities of the system. These can be people or things that interact with the system that is being modeled. For example, if we are modeling an online store we have many actors that interact with the store functionality. The customer browses the catalog, chooses items to buy, and pays for those items. A stocker will look at the orders and package items for the customer. A billing system will charge the customer's credit card for the amount purchase.
- Use Cases: Use cases are functional parts of the system. When we say what an actor does, that's a use case. The customer "browses the catalog", "chooses items to buy", and "pays for the items". These are all use cases. Many actors can share use cases. If we find a use case that is not associated with any actor, this may be a unnecessary functionality.
- Associations: Associations are shown between actors and use cases, by drawing a solid line between them. This only represents that and actor uses the use case.

There are also two kinds of relationships between use cases:

- Includes [3]: Use cases that are associated with actors can be very general. Sometimes they "include" more specific functionality. For example, the "pump gas" use case that is associated with the customer includes three use cases: Choose Gas Type, Fill Tank, and Calculate Total. Includes relationship is represented by dashed arrows that point to the included functionality. Beside the arrow is <<includes>>.
- Extends [3]: An extension use case is an insertion to the base use case. For example, some stores may allow for different payment options like credit card, debit card, or cash on delivery. These specific functionalities are extension of the general "pay for items." Extends relationship is represented by dashed arrows that point to the base functionality. Beside the arrow is <<extends>>

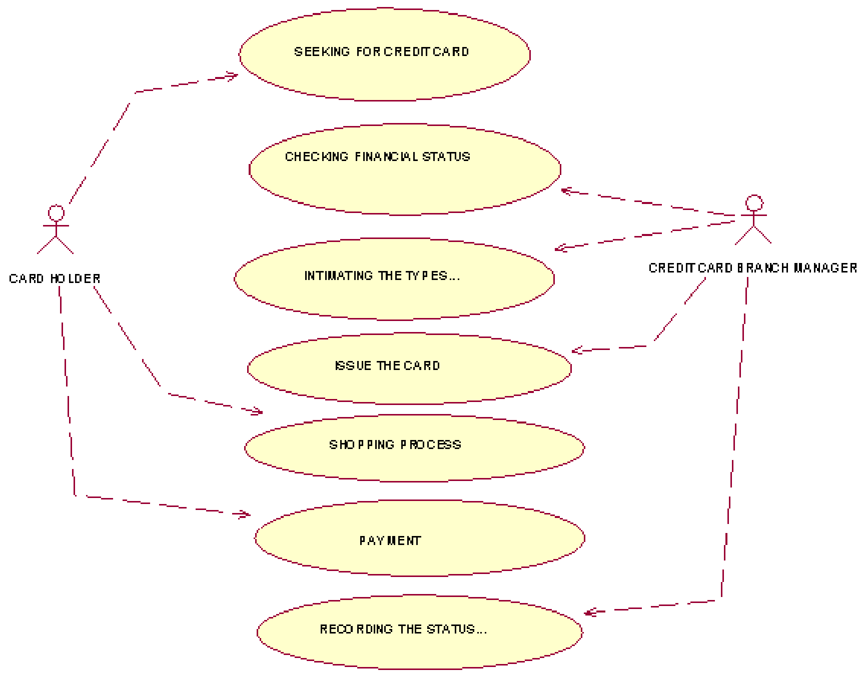

Fig 2. Use Case Diagram for Credit Card Management System

It is tempting to look at the structure of a use case document as a precursor or progenitor of the structure of the software it represents. However, this is not likely to be the case. Use cases are structured to minimize textual redundancy. While this is a laudable goal, it has nothing to do with software design considerations. Thus, the structure of the use cases is not related to the structure of the resultant software. Use cases do not represent objects or classes in the eventual designs. The relationships between use cases do not forshadow relationships in the software design. The structure of the use cases and the structure of the software are unrelated. Use case diagrams don't tell us very much. They convey the structure of the use cases, but tell us very little about the text within them. As such, they are not particularly interesting documents when they are separated from their textual descriptions. At best the diagrams provide a nice roadmap of relationships so that readers can reconstruct the whole text of a given scenario by tracing through the «uses» and «extends» relationships inserting the text of the former, and modifying the text according to the latter. Use cases are powerful tools for analysts to use when partitioning the functionality of a system. Use case relationships and the corresponding diagrams help analysts to structure use cases such that their textual descriptions contain a minimum of redundant information; thus making the whole text document much easier to maintain. But use cases are not design tools. They do not specify the structure of the eventual software, nor do they imply the existence of any classes or objects. They are purely functional descriptions written in a formalism that is completely separate from software design.

## IV. Interaction Diagram

Interaction diagrams [10] [11] [12] model the behavior of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams are sequence and collaboration diagrams. Interaction diagrams are used when you want to model the behavior of several objects in a use case. They demonstrate how the objects collaborate for the behavior. Interaction diagrams do not give a in depth representation of the behavior. Sequence diagrams, collaboration diagrams, or both diagrams can be used to demonstrate the interaction of objects in a use case. Sequence diagrams generally show the sequence of events that occur. Collaboration diagrams demonstrate how objects are statically connected.

ISSN(Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 4, Issue 2, February 2015**

The sequence diagram [3] [4] is used primarily to show the interactions between objects in the sequential order that those interactions occur. Much like the class diagram, developers typically think sequence diagrams were meant exclusively for them. However, an organization's business staff can find sequence diagrams useful to communicate how the business currently works by showing how various business objects interact. Besides documenting an organization's current affairs, a business-level sequence diagram can be used as a requirements document to communicate requirements for a future system implementation. During the requirements phase of a project, analysts can take use cases to the next level by providing a more formal level of refinement. When that occurs, use cases are often refined into one or more sequence diagrams. An organization's technical staff can find sequence diagrams useful in documenting how a future system should behave. During the design phase, architects and developers can use the diagram to force out the system's object interactions, thus fleshing out overall system design. One of the primary uses of sequence diagrams is in the transition from requirements expressed as use cases to the next and more formal level of refinement. Use cases are often refined into one or more sequence diagrams. In addition to their use in designing new systems, sequence diagrams can be used to document how objects in an existing (call it "legacy") system currently interact. This documentation is very useful when transitioning a system to another person or organization.
Diagram Elements:

- Object. Each of the objects that participate in the processing represented in the sequence diagram is drawn across the top. Note that objects are used in this diagram while classes are used in use cases, class diagrams, and state-transition diagrams.
- Lifeline. A dotted line is dropped from each object in the sequence diagram. Arrows terminating on the lifeline indicate messages (commands) sent to the object. Arrows originating on the lifeline indicate messages sent from this object to another object. Time flows from top to bottom on a sequence diagram.
- Active. To indicate that an object is executing, i.e., it has control of the CPU, the lifeline is drawn as a thin rectangle.
- Message. A horizontal arrow represents a message (command) sent from one object to another. Note that parameters can be passed as part of the message and can (optionally) be noted on the diagram.
- Return. When one object commands another, a value is often returned. This may be a value computed by the object as a result of the command or a return code indicating whether the object completed processing the command successfully. These returned values are generally not indicated on a sequence diagram; they are simply assumed. In some instances the object may not be able to return this information immediately. In this case, the return of this information is noted on the diagram later using a dotted arrow. This indicates the flow of information was based on a previous request.
- Conditional. Square brackets are used to indicate a conditional, i.e., a Boolean expression that evaluates to TRUE or FALSE. The message is sent only if the expression is TRUE.
- Iteration. Square brackets preceded by an asterisk (*) indicate iteration. The message is sent multiple times. The expression within the brackets describes the iteration rule.
- Deletion. An X is used to indicate the termination (deletion) of an object.

**Customer Signing in**

The customer, in order to perform online banking, contacts the banking site. The banking site asks for the information of the customer's account like the account number and the account details. After the customer has given the correct information, the banking computer calls the customer service department of the bank and the latter checks for the validation of this information. Then it grants/denies customer. After it has granted permission, the banking site asks for a user id and a password from the customer to perform transaction online.

After customer has given required information, the customer service department stores the entered information in the bank's database. The message between the bank's central computer and the customer service is synchronous because only after the response from the customer service department, the computer can send the needed information to the customer. The messages between the customer and the computer are asynchronous between the latter can or cannot make use of the options offered.

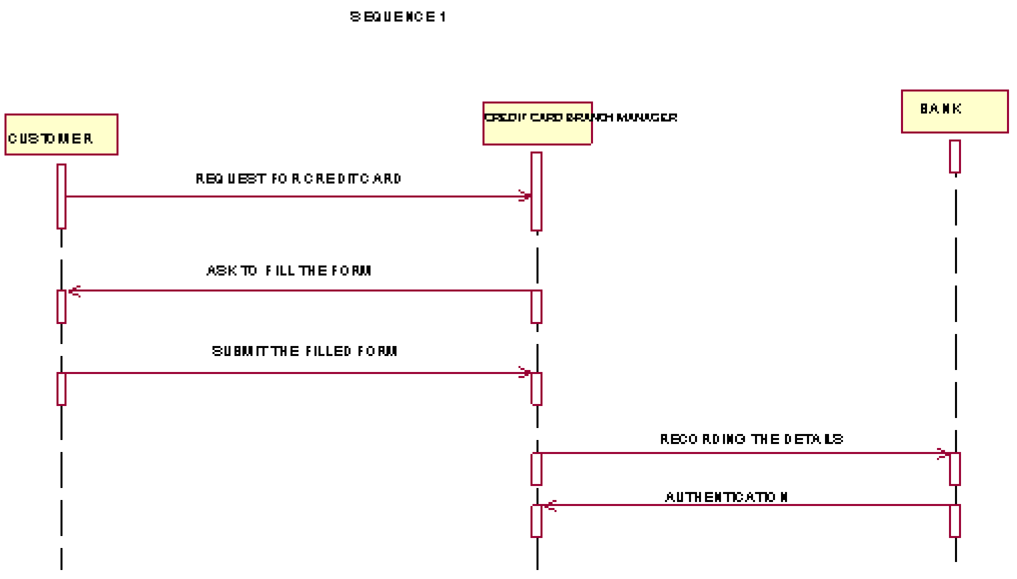

Fig 3. Sequence Diagram for Customer Sign in in Credit Card Management System

**Performing Transaction**
After the customer chooses the option for transaction, the banking site provides the option for the customer to perform transaction. Again, the customer gives the user id and password for validation by the customer service department and access is granted/ denied.

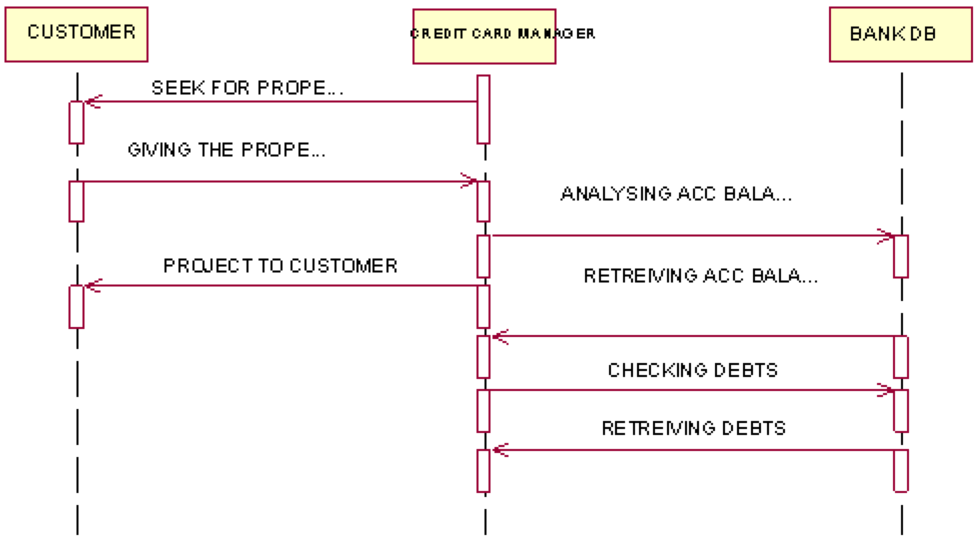

Fig 4. Sequence Diagram for Performing Transaction in Credit Card Management System

The customer service department checks for the minimum balance just in case if the customer wants to transfer funds between his account or making payments Always there will be an option for the customer to change his/her password

ISSN(Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 4, Issue 2, February 2015**

in the banking site
The customer can or cannot use this facility and if he/she uses this facility the new information is stored in the bank's database. Also the option for the "feedback" from the customer is always an option in the banking site, which the customer can or cannot respond. The customer can sign out he wishes. This is depicted in the Fig. 7and Fig. 8

**Making Payments**
The customer signs up and the Financial Institution checks for validation. The bank issues a bill to the customer to their email id. The customer makes the payment to the creditor. After the creditor, receives the payment, the financial institution issues receipts to the customer. The customer can provide the option to view the transaction status. The customer can sign out at this point and the financial institution suspends all the activities corresponding to the account. This is depicted in Fig. 9and Fig.10

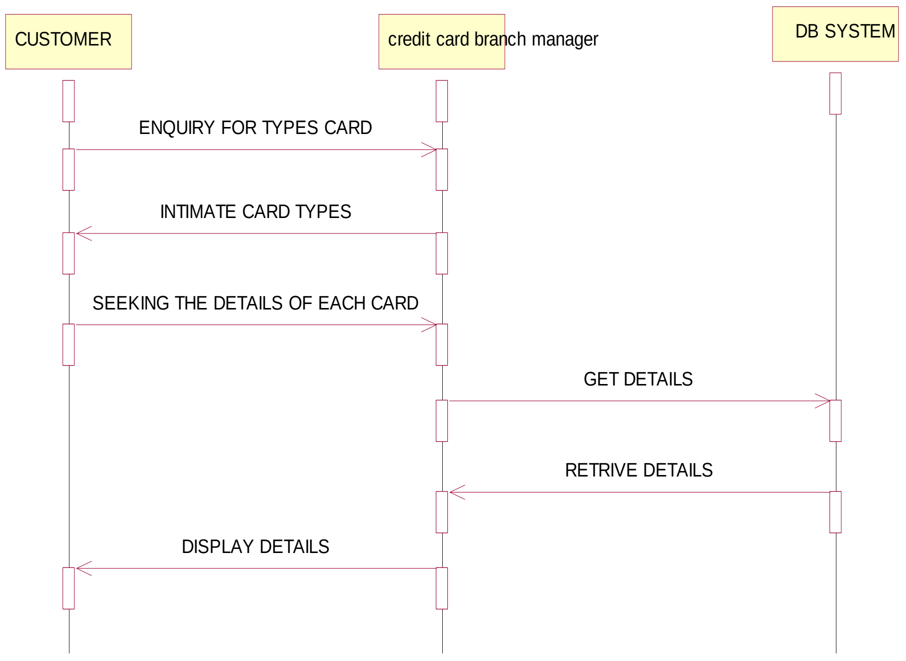

Fig 5. Sequence Diagram for Making Payments in Credit Card Management System

ISSN(Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 4, Issue 2, February 2015**

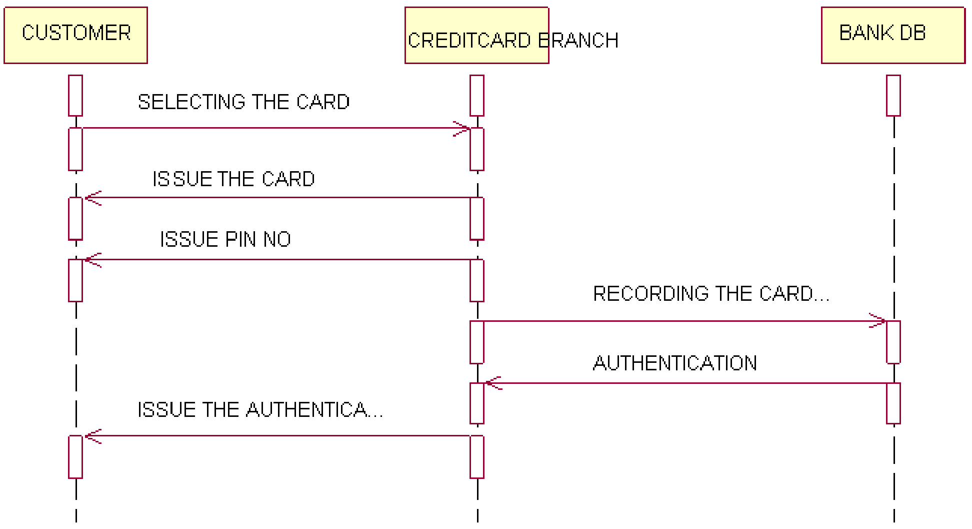

Fig 6. Sequence Diagram for Recording Transactions in Credit Card Management System

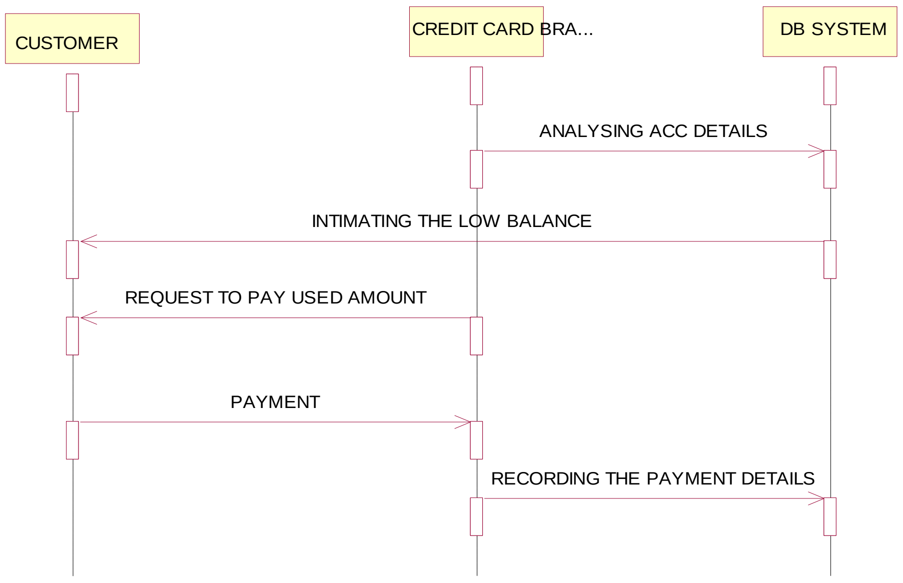
CUSTOMER
CREDIT CARD BRA...
DB SYSTEM
ANALYSING ACC DETAILS
INTIMATING THE LOW BALANCE
REQUEST TO PAY USED AMOUNT
PAYMENT
RECORDING THE PAYMENT DETAILS

Fig 6. Sequence Diagram for Intimation of Low Balance in Credit Card Management System

## V. Activity Diagram

UML 2 activity diagrams [3] [14] [15] are typically used for business process modeling, for modeling the logic captured by a single use case or usage scenario, or for modeling the detailed logic of a business rule. Although UML activity diagrams could potentially model the internal logic of a complex operation it would be far better to simply rewrite the operation so that it is simple enough that you don’t require an activity diagram. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) [5] from structured

development. Activity diagrams can show activities that are conditional or parallel. Activity diagrams should be used in conjunction with other modeling techniques such as interaction diagrams and state diagrams. The main reason to use activity diagrams is to model the workflow behind the system being designed. Activity Diagrams are also useful for: analyzing a use case by describing what actions need to take place and when they should occur; describing a complicated sequential algorithm; and modeling applications with parallel processes. However, activity diagrams should not take the place of interaction diagrams and state diagrams.
Activity diagrams do not give detail about how objects behave or how objects collaborate.
Building block elements in an Activity Diagram are:

- Initial Node

An initial or start node is depicted by a large black spot.

- Final Node

There are two types of final node: activity and flow final nodes. The activity final node is depicted as a circle with a dot inside. The flow final node is depicted as a circle with a cross inside. The difference between the two node types is that the flow final node denotes the end of a single control flow; the activity final node denotes the end of all control flows within the activity.

- Fork and Join Nodes:

Forks and joins have the same notation: either a horizontal or vertical bar (the orientation is dependent on whether the control flow is running left to right or top to bottom). They indicate the start and end of concurrent threads of control. A join is different from a merge in that the join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received. A merge passes any control flows straight through it. If two or more inflows are received by a merge symbol, the action pointed to by its outflow is executed two or more times.

- Decision and Merge Nodes:

Decision nodes and merge nodes have the same notation: a diamond shape. They can both be named. The control flows coming away from a decision node will have guard conditions which will allow control to flow if the guard condition is met.

## VI. Conclusion And Future Work

The paper has focused on the essential and extended Object Oriented features of an Online Personal Finance Management System. Various models of UML have been employed to analyze the static, dynamic and functional view of the system. In spite of several user-friendly, time-saving, cost-effective, convenient and flexible options provided by banks in the virtual space, Online Personal Finance has not achieved a major break-through that it deserves. However if an Object Oriented approach is adopted while building the software, an explosive growth in Online Personal Finance arena could be traced. Paper could be implemented using VB.net as front end and any reliable database system such as Oracle, as back end, and testing could be carried out by Rational Test Manager. The paper depicts the structural and behavioral aspects of online personal finance management system related to both logical and physical parts.. It is a fact that design flaws that surface during implementation are most costly to fix than those that are found earlier. Focusing on implementation issues too early restricts the design choices and often leads to inferior product. Hence the developed object-oriented approach encourages software developers to work and think in terms of application domain through most of the software engineering lifecycle. Since flaw in modeling process can substantially contribute to the development cost and time and affect the operational efficiency special attention is paid to the correctness of the UML models that are used at all planning levels rather than focusing much on implementation issues.

## References

1. Conrad Bock, U.S. National Institute of Standards and Technology, "UML 2 Composition Model", Journal of Object Technology, Vol.3, No.10, 2007
2. Thomas M.Atwood, "The Case for Object Oriented Databases" IEEE Spectrum, 0018-9235/91/ 0002-004, pp 44-48, 2007
3. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen "Object Oriented Modeling and Design", Prentice-Hall of India Private Limited.
4. Grady Booch, "Object-Oriented Analysis & Design with Applications", Pearson Education
5. Rogert.S.Pressman, "Software Engineering:A Practioner's Approach" Fifth Edition, McGraw-Hill Higher Education.

IJIRSET

ISSN(Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 4, Issue 2, February 2015**

6. Dionisis X. Adamopoulos and George Pavlou, University of Surrey, England Constantine A. Papandreou, Hellenic Telecommunications Organization (OTE) “Advanced Service Creation Using Distributed Object Technology”, IEEE Communications Magazine 0163-6804/02, 2007
7. http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/state.htm
8. http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm
9. http://www.agilemodeling.com/style/stateChartDiagram.htm
10. http://www.informit.com/articles/article.aspx?p=360441&seqNum=5
11. http://www.developer.com/design/article.php/3080941
12. http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=3718
13. http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_packagediagram.html
14. http://www.agilemodeling.com/artifacts/activityDiagram.htm
15. http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_activitydiagram.html
16. http://www.agilemodeling.com/artifacts/useCaseDiagram.htm
17. http://www.developer.com/design/article.php/2109801
18. http://www.andrew.cmu.edu/course/90-754/umlucdfaq.html
19. http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/use_case.htm
20. http://www.objectmentor.com/resources/articles/usecases.pdf
21. http://www.agilemodeling.com/artifacts/classDiagram.htm
22. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/
23. http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/class.htm
24. http://www.developer.com/design/article.php/2206791