Modeling and Validation of Object-Oriented Database System

Vipin Saxena and Ajay Pratap, *Member, IACSIT*

***Abstract*—The present paper deals with the role of Unified Modeling Language (UML) in the object-oriented database system. The object oriented database technology is considered as fifth generation database technology and it is a combination of object-oriented system and the database management system. A real case study of Indian Postal Services (Savings Bank) is discussed i.e. how various banking services and products can be classified. The db4o is an open source object-oriented database management system, which is used to store the data and its modeling aspects, which are represented with the help of UML use case, class, state and sequence diagrams. Validation of the state diagram is done by transforming the UML state diagram into finite state machine and valid input strings are generated as test cases.**

***Index Terms*—Class diagram, finite state machine, oodbms, sequence diagram, state diagram, uml, use-case diagram and validation**

I. Background and Related Work

An object oriented database integrates object orientation along with databases. If a Database Management System (DBMS) supports the object oriented data model then it is called as Object-Oriented Database Management System (OODBMS). In the current scenario, there are many object-oriented database products which are available in the market like O2, GemStone, ONTOS, ObjectStore, Ozone, Objectivity/DB and db4o. OODBMS began developing in the mid-80's but due to the lack of maturity of early days products, OODBMS could not get wide acceptance in the market. Many of them were not full-fledged database systems as compare to the relational database systems [1]. Let us briefly explain the status of object-oriented database in the current scenario and support of UML to represent the object-oriented database as many of the software Industries are shifting the old structured database into the object-oriented database.

One of the important reference paper [1] described about the object oriented schema and the techniques to make it more expressive. In the recent years, the UML has emerged as the defacto standard for the representation of software engineering diagrams and Gibson [2] discussed the various factors for the implementation of object oriented database system. Meilir [3] also described the fundamentals issues of object-oriented design in the UML alongwith various kinds of diagrams used to represent UML models. In [4], the conditions for using the object-oriented database system are explained and author also discussed the weaknesses of the object-oriented system. In [5], Standard Object Modeling Language is described and [6] helps the researchers for the designing of various kinds of UML diagrams like use case, class, sequence, activity and state diagrams, etc. The architecture of object-oriented database is also discussed in [7] and it describes the two-tier and three-tier architectures. Object-oriented systems and its designing with the UML are presented in [8]. In [9], UML design approach for the object-oriented database system is discussed.

Hierarchical approach is used to model large web applications. This approach builds hierarchies of Finite State Machines (FSM) that model subsystems of the web applications and then generates test requirements. Andrews et al [10] introduces a prototype implementation to support the technique. Hopcroft et al [11] presented the basics of automata theory and also discussed some advanced concepts with case studies.

Theory of finite automata is used to understand the structure, behavior, and limitations of logic machines. Kohavi and Jha [12] discussed the logical design and testing with the help of examples. Finite state machines are widely used to model systems in diverse areas, including sequential circuits, certain types of programs, and, more recently, communication protocols. Lee and Yannakakis [13] discussed the fundamental problems in testing finite state machines and techniques for solving these problems. It is not possible to apply rigorous automated analysis or to execute a UML model in order to test its behavior, short of writing code and performing exhaustive testing. McUmber and Chang [14] introduced a general framework for formalizing a subset of UML diagrams in terms of different formal languages based on a homomorphic mapping between meta-models describing UML and the formal language.

Due to lack of representation of object-oriented databases through UML, the present paper is an attempt to develop an object-oriented system for Indian Postal System [15] which describes various kinds of the saving bank services and it also deals with various rules and conditions. Object-oriented data modeling is done to store large amount of databases. By the use of UML, use case, class, sequence and state diagrams are designed and these diagrams give pictorial representation of the real case study. The state diagram is transformed into FSM and its equivalent grammar is generated. The production rules are used for the validation of proposed UML model and various test cases are generated to validate the proposed UML model.

II. Object-Oriented Data Model

Modeling is an abstraction of something for the purpose of understanding it before building it. Object-oriented data model (OODM) represents that anything in the world can be represented by only one modeling concept i.e. through object.

Manuscript received August 22, 2012; revised September 27, 2012.

Vipin Saxena is with the Deptt. of Computer Science (e-mail: vsax1@rediffmail.com)

The OODM is based on a number of basic concepts, namely object, class, abstraction, encapsulation, inheritance and polymorphism.

Object-oriented data model in comparison of relational database model is shown below in fig. 1.

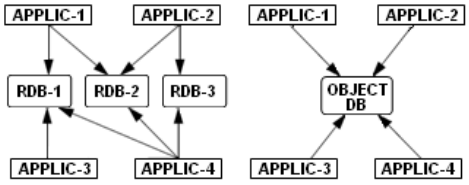

Fig. 1. Relational database model vs object-oriented data model

Let us explain some of the fundamental concepts in brief:

Object: Each object has its well defined state, behavior and unique identification known as object identifier (OID). State is the current value of its attributes and behavior which tells how an object reacts when any operation is performed on object. The information may be organized in graphs of objects. Objects are the instances of classes and the classes are related to each other by inheritance.

Class: Objects with the same properties and behavior are grouped into classes and the object can be an instance of only one class or of several classes.

Abstraction: It is the process of identifying the key aspects of an entity and ignoring the rest.

Encapsulation: Hiding the data members of an object from an object from the outside world is known as encapsulation.

Inheritance: It is a property of a class hierarchy where the subclass inherits attributes and methods of its super-class.

Polymorphism: It is the ability of different objects to respond to the same message in different ways.

The Unified Modeling Language is a language to specify, construct, present and document the artifacts of software system. The usage of UML helps to minimize the need to write long documentation and it is a modeling language which is the most general methods for developing software. In the recent years, the UML has emerged as the defacto standard for the representation of software engineering diagrams. Modeling through Unified Modeling language has three categories: Class Model, State Model and Interaction Model. The complete class model is designed with the help of objects and good classes while the state model is the representation of states and various kinds of events and the interaction model are prepared with the help of activity, use case, and sequence diagram.

III. Modeling of the System Through UML

Modeling is a basic unit for the development of any system. The model represents the system, promotes the understanding and enables the simulations. Usually, the models are described by visual languages i.e. most of the information in the model is expressed by graphical symbols and connections. Previously, Indian Post Office was storing the data manually. Later on the computerization process started but the database, which was being used, was relational. Since the amount of data in post office is very large and the due to variety of the services and the need of various types of data types, the complexity also increases. Therefore, it is a big challenge to handle large amount of database & for this purpose, object-oriented model is designed by the use of UML which is described below in brief.

A. Use Case Diagram

The use case is a piece of functionality that the system can provide by interacting with the actors. A use case diagram consists of actors and the use cases. The actors are the direct external user of the system. In other words one can say that it is an object or the set of objects that communicates directly with the system but that is not the part of system. The diagram involves a sequence of messages between the use cases and the actors.

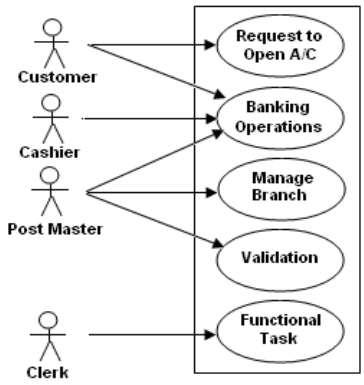

Fig. 2. UML use case diagram for post office banking

Fig. 2 shows the use case diagram of the saving bank's account opening event. The interaction between the four actors named as Customer, Cashier, Post_Master & Clerk are shown above with the five use cases performed by them. Customer sends the request to the branch of post office for opening the account and interacts with the actor clerk. The clerk is involved in various functional tasks such as accepting forms, handling the payment counter and sale of other postal products. Final approval for opening the account of customer is given by the post master after performing checks at different levels. After the validation, the customer deposits the amount to the cahier. The post master is involved in banking operations and management of the branch.

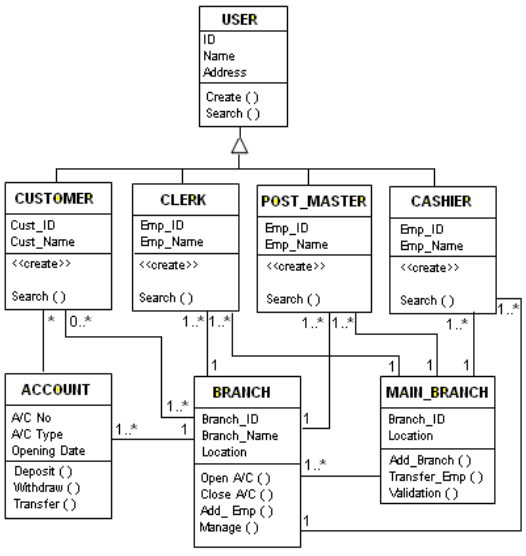

Fig. 3. UML class diagram for post office banking

B. UML Class Diagram

The UML class diagram shows the static structural behavior of the system, in which attributes and operations are designed for the complete system. The classes can be related to each other in number of ways, like they can be associated, dependent, specialized or packaged. A system can have a number of class diagrams because not all classes participate in a single class diagram. Fig. 3 depicts the class diagram of the banking system. The class diagram has eight persistent classes, which are USER, CUSTOMER, CLERK, POST_MASTER, CASHIER, ACCOUNT, BRANCH and MAIN_BRANCH. These classes are connected to each other by various relationships with their multiplicities as shown in the diagram. CUSTOMER, CLERK, POST_MASTER and CASHIER are subclasses and they are inherited from non-abstract class USER.

C. UML Sequence Diagram

Sequence diagram tells how objects interact with each other i.e. how messages are being send and received between objects. This diagram has two vertices: The vertical axis shows time and known as lifeline. Each actor as well as the system is represented by the vertical line. The horizontal axis shows the messages, which are being sent from the sender to the receiver. The sequence diagram for the savings bank scheme of post office system is shown in fig. 4.

Each use-case requires one or more sequence diagrams to describe its behaviour. It is the best approach to show a specific portion of the use case and one sequence diagram for each major flow of control.

The sequence diagram presented in Figure 4 shows that the customer object sends request for opening an account to the clerk object. Validation is performed by the clerk and the post master. These validity checks may be the eligibility for opening account, minimum money to be deposited, address proof, age proof, etc. After that, it is forwarded to the post master for the final approval. The response generated by the post master is sent to the customer.

Finally, at least the minimum amount for opening an account is deposited by the customer to the cashier. The cashier enters the data related to the customer and the account number is generated. A pass-book is also issued to the customer.

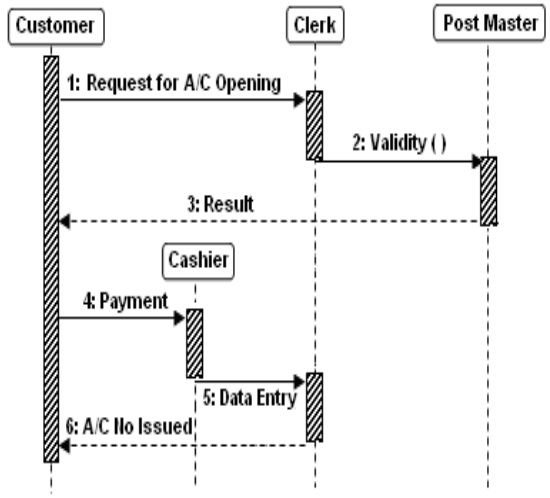

Fig. 4. UML sequence diagram for opening account

IV. Validation of UML Model

A. UML State Diagram

UML state diagram is a graph whose nodes are states and directed arcs are the transitions between them. They are used to describe the behaviour of a system. It describes all of the possible states of an object as events occur. The state model of any system consists of multiple state diagrams. The state diagram must match on their interfaces: events and guard conditions. The individual state diagrams interact by passing events and through the side effects of guard conditions. A state is a condition during the life of an object which satisfies some condition, performs some activity, or waits for some external event.

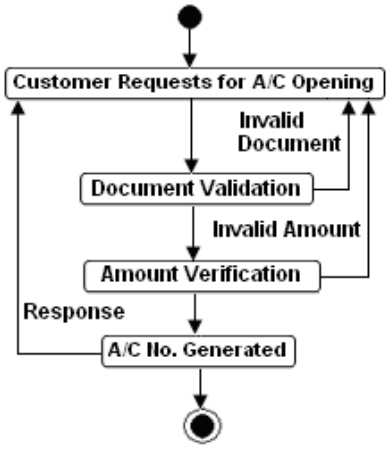

Fig. 5 (a). UML state diagram for account opening

Fig.5 [a] depicts the state diagram of the system for opening an account by the customer. The account number is generated after the document validation and amount verification.

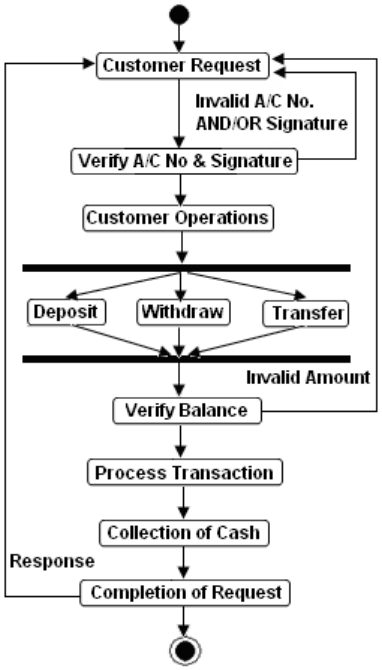

Fig. 5 (b). UML state diagram for banking operations

Fig. 5 [b] depicts the banking operations performed by the customer. After checking the account number and verifying

the signature from the database, the customers may choose the banking operations. These operations are deposit, withdrawal and transfer of money. The withdrawal case is taken for the discussion.

B. Basics of Finite-State Machine

In finite state machines the number of possible states and the number of inputs both are finite and the change of the state is totally governed by the inputs. Implementation of FSM can be done by formal languages. The theory of formal languages is an area, which provides a firm basis for development of number of applications in computer science.

A finite state automation is represented as $(Q, \Sigma, \delta, q_0, F)$ where Q is finite non empty set of states, Σ is finite non empty set of inputs, δ is function which maps $Q \times \Sigma$ into Q and known as direct transition function, q_0 ($q_0 \in Q$)is initial state and F ($F \subseteq Q$) is the set of final states. There may be one or more than one final states.

A grammar is required for the formation of sentences in any language which is defined as (V_N, Σ, P, S) where V_N is a finite nonempty set whose elements are called variables, Σ is a finite nonempty set whose elements are called terminals and $V_N \cap \Sigma$, S is a special variable called the start symbol and belongs to V_N . In this grammar P is a finite set whose elements are $\alpha \rightarrow \beta$, where α and β are strings on $V_N \cup \Sigma$. Elements of P are called production rules.

C. Transformation of UML State Diagram into FSM

The state diagrams present in fig. 5(a) and fig. 5(b) are transformed into finite state machine shown in fig. 6(a) and fig. 6(b). In fig. 6(a), we have assumed the customer's end as initial state and final state, which is equivalent to state q_0. When the user requests for opening account, then the required documents are checked with input {a} and the state is changed to q_1. But if the documents are not valid then the input {a'} sends it in the previous state q_0 with a message "Invalid Document". Now the system verifies the minimum amount with input {b} and it changes its state to q_2 but if the amount is not fulfilling the criteria for minimum balance amount for opening the account the input {b'} sends it to the initial state q_0. After the document validation and amount verification, the account number is generated and the input {c} sends it in the state q_3 and the account number is informed to the customer and the system goes in the initial state q_0 when input {d} is given to the system. State diagram present in fig. 5(a) is transformed into transition graph shown bellow in fig. 6(a) and Table I presents the transition table related to the transition graph.

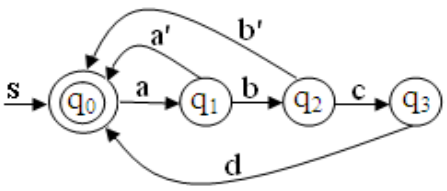

Fig. 6 [a]: Transition graph for Opening Account

Fig. 5 (b) is transformed into transition graph, which is present in fig. 6(b) and Table II presents the transition table related to the transition graph

In fig. 6(b), the customer's end is initial state as well as final state, which is equivalent to state q_0. The customer requests for banking operations, then the customer's account number and signature are verified. The correct account number and signature is given by input {a} and the system turns into in state q_1. But the wrong account number and/or signature will convert the system go in the initial state q_0. The machine reaches in state q_2 from q_1 after getting the input {b}. Input {b} shows the customer's request for banking operations. The system goes in state q_3, q_4 and q_5 by getting the inputs {c}, {d} and {e} according to the selection of operation by the customer. The change in state q_3, q_4 and q_5 are due to operations transfer, withdrawal and deposit. The value of amount say input {f} changes their state to q_6.

TABLE I: TRANSITION TABLE FOR OPENING ACCOUNT

δ/Σ	A	a'	b	b'	c	D
→q_0	q_1	Φ	Φ	Φ	Φ	Φ
q_1	Φ	q_0	q_2	Φ	Φ	Φ
q_2	Φ	Φ	Φ	q_0	q_3	Φ
q_3	Φ	Φ	Φ	Φ	Φ	q_0

Here we are taking the case of withdrawal operation, which is valid only if after withdrawing the requested amount minimum balance is left in the customer's account. In this situation the system gets the input {g} and reaches to state q_7 for processing of transaction otherwise reaches to the initial state after getting input {f '}if the requested amount is not valid. After successful completion, the system gets input {h} and goes in state q_8 where the customer receives the cash. Finally the system reaches in state q_9 after getting the input {i} as a completion of request and a response is transfered in the form of input {j} to the customer's end, which is final state as well as initial state.

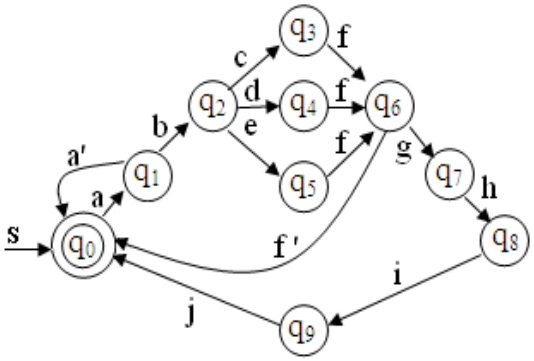

Fig. 6 (b). Transition graph for banking operations

TABLE II: TRANSITION TABLE FOR BANKING OPERATIONS

δ/Σ	a	a'	b	c	d	e	f	f '	g	h	i	j
→q_0	q_1	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ
q_1	Φ	q_0	q_2	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ
q_2	Φ	Φ	Φ	q_3	q_4	q_5	Φ	Φ	Φ	Φ	Φ	Φ
q_3	Φ	Φ	Φ	Φ	Φ	Φ	q_6	Φ	Φ	Φ	Φ	Φ
q_4	Φ	Φ	Φ	Φ	Φ	Φ	q_6	Φ	Φ	Φ	Φ	Φ
q_5	Φ	Φ	Φ	Φ	Φ	Φ	q_6	Φ	Φ	Φ	Φ	Φ
q_6	Φ	Φ	Φ	Φ	Φ	Φ	Φ	q_0	q_7	Φ	Φ	Φ
q_7	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	q_8	Φ	Φ
q_8	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	q_9	Φ
q_9	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ	q_0

D. Generation of Equivalent Grammar

Transition functions and equivalent grammar for opening the account are given bellow,

$\delta(q_0, a) = q_1 \Rightarrow q_0 \rightarrow a\ q_1$
$\delta(q_1, a') = q_0 \Rightarrow q_1 \rightarrow a'\ q_0$
$\delta(q_1, b) = q_2 \Rightarrow q_1 \rightarrow b\ q_2$
$\delta(q_2, b') = q_0 \Rightarrow q_2 \rightarrow b'\ q_0$
$\delta(q_2, c) = q_3 \Rightarrow q_2 \rightarrow c\ q_3$
$\delta(q_3, d) = q_0 \Rightarrow q_3 \rightarrow d\ q_0$

Transition functions and equivalent grammar for banking operations are given bellow,

$\delta(q_0, a) = q_1 \Rightarrow q_0 \rightarrow a\ q_1$
$\delta(q_1, a') = q_0 \Rightarrow q_1 \rightarrow a'\ q_0$
$\delta(q_1, b) = q_2 \Rightarrow q_1 \rightarrow b\ q_2$
$\delta(q_2, c) = q_3 \Rightarrow q_2 \rightarrow c\ q_3$
$\delta(q_2, d) = q_4 \Rightarrow q_2 \rightarrow d\ q_4$
$\delta(q_2, e) = q_5 \Rightarrow q_2 \rightarrow e\ q_5$
$\delta(q_3, f) = q_6 \Rightarrow q_3 \rightarrow f\ q_6$
$\delta(q_4, f) = q_6 \Rightarrow q_4 \rightarrow f\ q_6$
$\delta(q_5, f) = q_6 \Rightarrow q_5 \rightarrow f\ q_6$
$\delta(q_6, f') = q_0 \Rightarrow q_6 \rightarrow f'q_0$
$\delta(q_6, g) = q_7 \Rightarrow q_6 \rightarrow g\ q_7$
$\delta(q_7, h) = q_8 \Rightarrow q_7 \rightarrow h\ q_8$
$\delta(q_8, i) = q_9 \Rightarrow q_8 \rightarrow i\ q_9$
$\delta(q_9, j) = q_0 \Rightarrow q_9 \rightarrow j\ q_0$

From the above grammar, one can generate some test cases with the help of grammar and it can be verified from the production rules.

Test Case 1: The clerk at the branch will verify the account number and the signature from the database and if any of them is wrong then a message is passed to the customer. From stage q_0 to q_1 an input {a} is given and in case of wrong account number or signature mismatch the input {a'} transfers it in stage q_0.

$$q_0 \rightarrow a\ q_1$$

$$q_1 \rightarrow a'\ q_0$$

By replacing the non terminals on RHS of productions, we get

$$q_0 \rightarrow a\ a'\ q_0 \quad (1)$$

Derived production (1) shows that the wrong account number or signature will send the system in the initial state. Only for the correct input, the system will proceed.

Test Case 2: After the verification of account number and signature, the clerk will verify the withdrawal amount. In fig. 5 (b), three operations and case of withdrawal are taken for analysis. The amount verification can be done from the following productions:

$$q_4 \rightarrow f\ q_6$$
$$q_6 \rightarrow f'q_0$$
$$q_6 \rightarrow g\ q_7$$

By replacing the non terminals on RHS of productions, we get

$$q_4 \rightarrow f\ g\ q_7 \quad (2)$$

$$q_4 \rightarrow f\ f'q_0 \quad (3)$$

Derived production (2) shows that if the withdrawal amount is valid then the operation will proceed successfully and the possible inputs string is "f g". Production (3) shows that if the withdrawal amount is not valid i.e. after subtracting the withdrawal amount from the current balance, minimum balance is not available then the system will come to the initial state.

Test Case 3: For processing the withdrawal operation the possible number of inputs can be drawn from the following productions:

$$q_0 \rightarrow a\ q_1$$
$$q_1 \rightarrow b\ q_2$$
$$q_2 \rightarrow d\ q_4$$
$$q_4 \rightarrow f\ q_6$$
$$q_6 \rightarrow g\ q_7$$

By replacing the non terminals on RHS of productions, one can get

$$q_0 \rightarrow abdfg\ q_7 \quad (4)$$

Derived production (4) shows that for the successful withdrawal operation input string is "abdfg" is required.

V. Conclusions and Future Scope of Work

In the present work, an object oriented data model has been proposed to implement the post office savings bank scheme. Modeling is done with the help of UML diagrams shown as use case diagram, class diagram, sequence diagram and state diagram. Nowadays the banking system is using relational database management system but using the proposed UML model one can also use the Object Oriented database management system, which is more powerful in complex situation and queries. Validation of proposed UML model is done by transforming it into the finite state machine and generating the various production rules. Further one can perform transformation of few complex modeling features such as qualifiers, association classes, compositions and generalizations for the ease of use.

Acknowledgment

The authors are grateful to Prof. B. Hanumaiah, Vice-Chancellor, Babasaheb Bhimrao Ambedkar University (A Central University), Lucknow, India for providing the excellent facility in the computing lab of University.

References

[1] D. Calvanese and M. Lenzerini, "Making Object-Oriented Schemas More Expressive," in *Proc. 13th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems* (PODS'94), Minneapolis, MN, USA, 1994.
[2] B. Gibson, "A Case Study of Object Database Implementation," *APM, 1608.00.02, Technical Report*, 10th October, 1995.
[3] P. J. Meilir, "Fundamentals of Object-Oriented Design in UML," 2*nd ed. Boston, Addison-Wesley*, 2005.
[4] S. Bagui, "Achievements and Weaknesses of Object-Oriented Databases," *Journal of Object Technology*, vol. 2, no. 4.
[5] F. Martin, "UML Distilled Third Edition: A Brief Guide to the Standard Object Modeling Language," *3rd ed. Boston*, Addison-Wesley, 2004.
[6] J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language Reference Manual," *2nd ed. Addison-Wesley*, Boston, 2005.

[7] S. Luthra, "Architecture in Object Oriented Databases," in *Proc. The National Conference on Challenges and Opportunities in Information Technology* (COIT-2007), IET- Mandi Govindgarh, Punjab, 2007.
[8] L. V. Tagliati and C. Caloro, "UML and Object Oriented Drama," *Journal of Object Technology*, vol. 7, no. 1.
[9] U. A. Umoh, E. O. Nwachukwu, I. J. Eyoh, and A. A. Umoh, "Object Oriented Database Management System: A UML Design Approach," *The Pacific Journal of Science and Technology*, vol. 10, no. 2, November 2009.
[10] A. A. Andrews, J. Offutt and R. T. Alexander, "Testing web applications by modeling with FSMs," *Software and Systems Modeling*, vol. 4, no. 3, pp. 326-345, July 2005.
[11] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory," *Languages and Computation, 2nd edition, Boston*, Addison-Wesley, 2001.
[12] Z. Kohavi and N. K. Jha, "Switching and Finite Automata Theory," *3rd edition, Cambridge University Press*, 2009.
[13] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines - A Survey," in *Proc. of the IEEE*, vol. 84, no. 8, 1996, pp. 1090-1126.
[14] W. E. McUmber and B. H. C. Cheng, "A General Framework for formalizing UML with formal Languages," in *Proc. 23rd International Conference on Software Engineering*, Toronto, Canada, 2001, pp. 433-442.
[15] India Post. [Online]. Available: www.indiapost.gov.in/

Vipin Saxena is a Professor & Head, Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow, India. He got his M.Phil. Degree in Computer Application in 1991 & Ph.D. Degree work on Scientific Computing from University of Roorkee (renamed as Indian Institute of Technology, Roorkee, India) in 1997. He has more than 16 years teaching experience and 19 years research experience in the field of Scientific Computing & Software Engineering. Currently he is proposing various software designs by the use of Unified Modeling Language for the research problems related to the Software Domains & Advanced Computer Architecture. He has published more than 91 International and National research papers in various refereed Journals and authored four books covering Software Engineering, E-Learning and Operating System. Dr. Saxena is a life time member of Indian Congress.
Mobile: 0091-9452372550

Ajay Pratap got M.Phil. Degree in Computer Science in 2008 and Master of Computer Application in 2006.He is a research student in Department of Computer Science, Babashaheb Bhimrao Ambedkar University, Lucknow, India. Presently, he is working on performance evaluation of object oriented database through UML.