European Journal of Applied Sciences 9 (4): 212-218, 2017
ISSN 2079-2077

DOI: 10.5829/idosi.ejas.2017.212.218

Smart Healthcare: A New Paradigm

[1]S. Balamurugan, [2]Nikita Mary Ablett, [3]V. Mohanapriya and [4]B. Powmeya

[1]Department of CSE, KIT- Kalaignarkarunanidhi Institute of Technology, Coimbatore, India
[2]Software Engineer, Infosys Pvt. Ltd., Bengaluru, Karnataka, India
[3]Software Engineer, Cognizant Technology Solutions, Pvt. Ltd., Coimbatore, India

Abstract: Recent years have observed a steep rise in substantial financial resources spent against health care sector. Ensuring a high operational efficiency in the sector turns out to be the most essential goal for evaluating the organizational performance as a whole. This work aims to employ a modeling language aimed to provide a unified framework for representing an effective and efficient health care management system, which implicitly ensures the privacy and confidentiality of the information and messages. The proposed modeling methodology is based on the principle of object orientation, which allows describing both software and functionalities explicitly. 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 healthcare database 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) takes its impeccable role.

Key words: Health Care Sector · Unified Framework · Privacy · Confidentiality · Unified Modeling Language · Semantics

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,

Corresponding Author: S. Balamurugan, Department of CSE, KIT- Kalaignarkarunanidhi Institute of Technology, Coimbatore, India.

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 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.

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>>

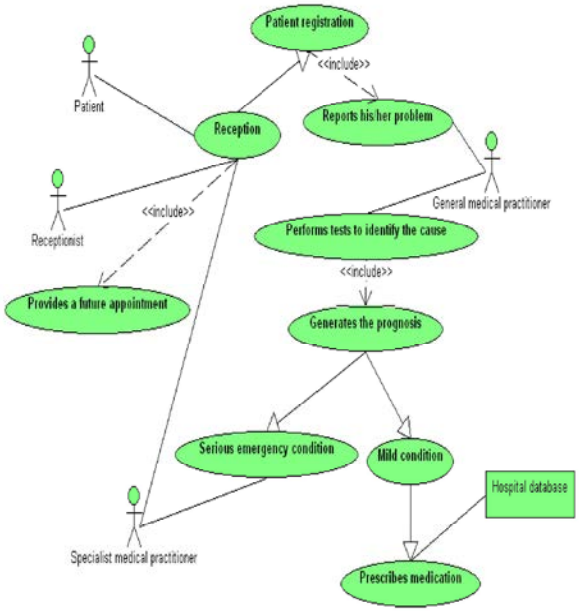

Fig. 1: User case diagram for hospital management

The above use case diagram gives the overview of hospital management system.when patient goes to a hospital, he would be requested to fill in registration form. He would have to provide details such as the patient's name, age, residential address, contact information and his previous medical history, if any to the receptionist. The hospital maintains a file for each patient which contains information about the patient's medical history, the results of various tests undergone and the dates of appointments with the concerned doctors which would be updated after each visit. The patient would be directed to a general medical practitioner. He performs various tests and identifies the problem. If the problem is in mild state, the patient is advised to undergo medication. In case of serious conditions, the patient is referred to a specialist medical practitioner who would conduct further tests to ascertain the exact cause of the problem. The condition of every patient is kept up-to-date in the hospital database.

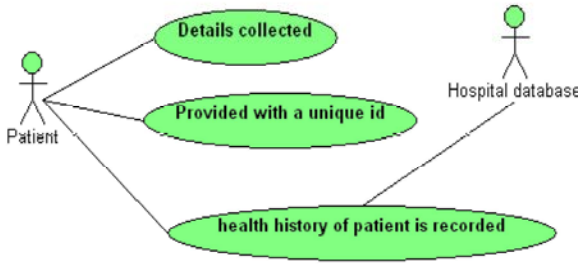

Fig. 2: Use case diagram for patient registration

Initially when a patient goes to a hospital, he would be requested to fill in the registration form. He would have to provide details such as the patient's name, age, residential address, contact information and his previous medical history, if any. Then the patient would be provided with a registration card containing information like name, registration number, age and sex of the patient. The hospital also maintains a file for each patient which contains information about the patient's medical history, the results of various tests undergone and the dates of appointments with the concerned doctors which would be updated after each visit.

After the patient registration, he/she is provided an appointment with the concerned doctor. The doctor conducts a detailed investigation of the patient. Then, based on the seriousness of the diagnosis the doctor either recommends a surgery or prescribes medicines to the patient. The patient is then advised to report for follow up checkups in order to assess the progress of the patient.

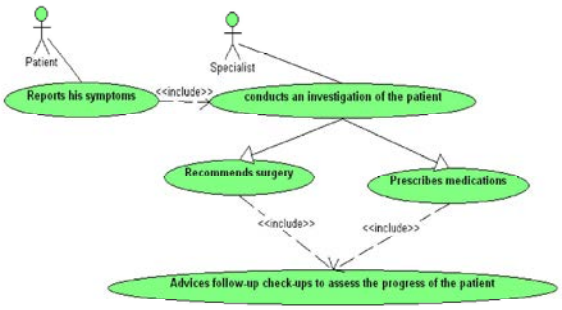

Fig. 3: Use case diagram of the diagnosis procedure

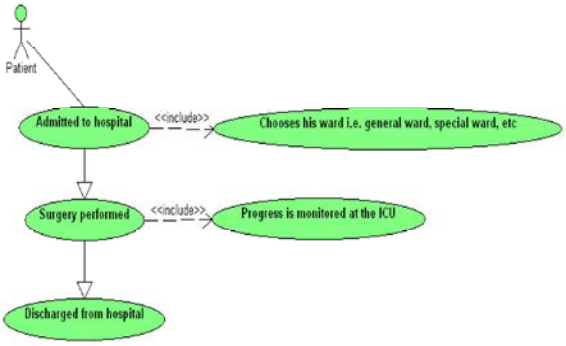

Fig. 4: Use case diagram for a patient undergoing surgery

After the diagnosis if the patient is recommended a surgery then he would be asked to get admitted to the hospital. Upon admission, the patient would get to choose the ward in which he would stay based on his individual preferences. According to the availability of the operation theatre and the surgeon a suitable date and time is fixed for the surgery. After the surgery the patient is moved to the Intensive Care Unit (ICU) where the vital parameters of the patient are monitored. He is then moved to his ward and kept under observation for a few more days and then discharged.

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.

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.

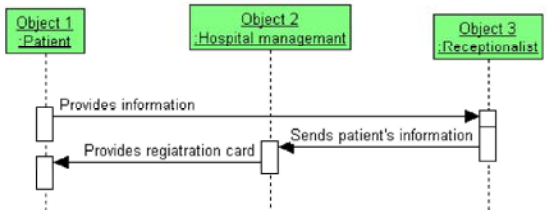

Fig. 5: Sequence diagram for patient registration

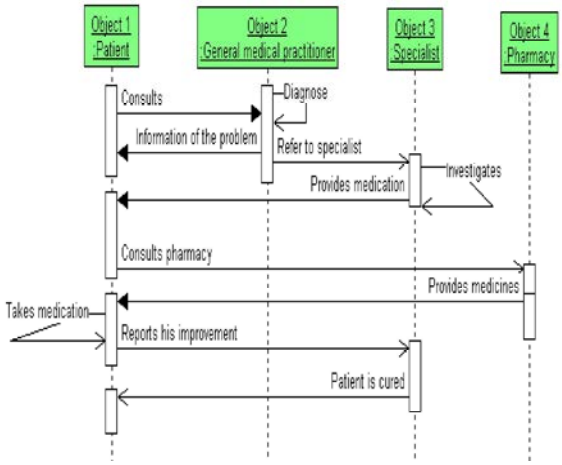

Fig. 6: Sequence diagram for medication process

The patient in order to get treatment from the hospital, contacts the reception for registration. The receptionist gets the information of the patient like name, address and the patient's problem. After the patient has

given all the required information, the receptionist sends the information to the hospital management. The hospital management allocates a unique id to the patient and also provides the patient with a registration card.

The patient consults the general medical practitioner who diagnoses the problem and advises for the further treatment. In case of critical condition, the patient is referred to a specialist medical practitioner..The specialist, further investigates about the problem and prescribes medication to the patient. The patient acquires the required medication from the pharmacy. Upon consumption of these medications,the patient is cured of his problem.

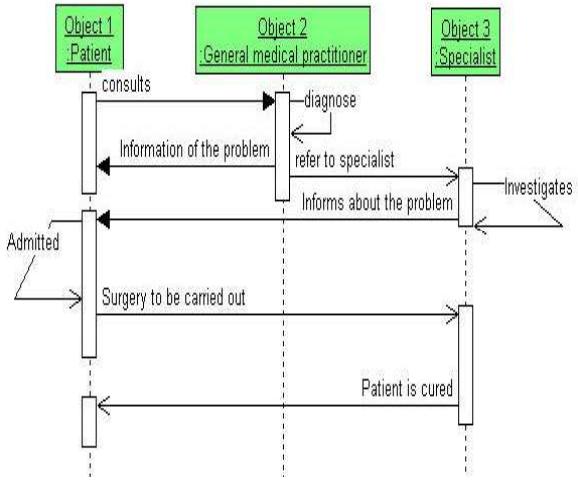

Fig. 7: Sequence diagram for undergoing surgery

The patient consults the general medical practitioner who diagnoses the problem and advises for the further treatment.In case of critical condition,the patient is reffered to a specialist medical practitioner..The specialist, further investigates about the problem and advises surgery for the speedy recovery of the patient. The patient admits himself in the hospital and he is prepared for the surgery. He undergoes surgery and is kept in the post operative care for a few hours and then transferred to his ward. He is kept under observation for a few days and then discharged.

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.

When a patient goes to a hospital, he is requested to fill in the registration form. Then the patient would be provided with a registration card. The hospital maintains a file for each patient which contains information about the patient's medical history, the results of various tests undergone and the dates of appointments with the

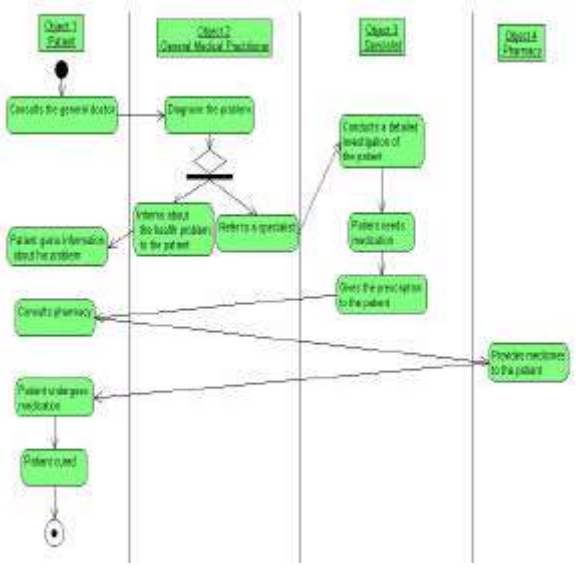

Fig. 8: Activity Diagram for a patient to undergo medication

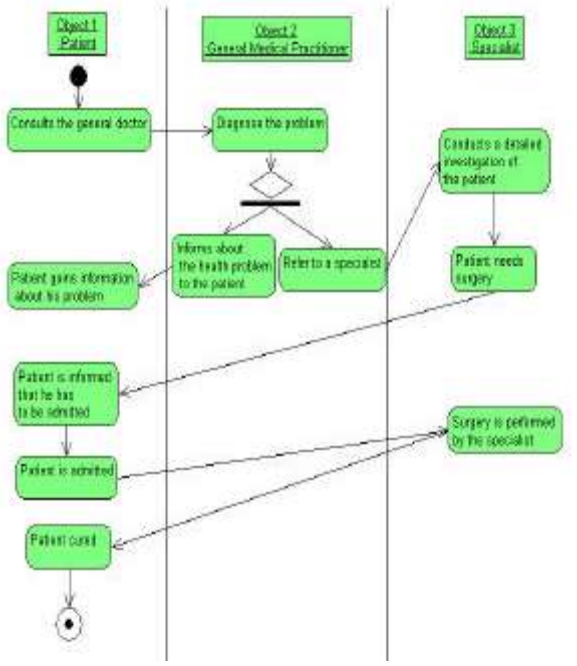

Fig. 9: Activity Diagram for a patient to undergo surgery

concerned doctors which would be updated after each visit. After the patient registration, he/she is provided an appointment with the general medical practitioner who conducts a detailed investigation of the patient. Then, based on the seriousness of the diagnosis the doctor either refers the patient to a specialist or prescribes medication. The patient then acquires medications from the pharmacy. He is then advised to report for follow up check up in order to assess his progress.

After consultation with the doctor if a surgery is advised then the patient would be asked to get admitted to the hospital. Upon admission, the patient would get to choose the ward in which he would stay based on his individual preferences. According to the availability of the operation theatre and the surgeon a suitable date and time is fixed for the surgery. The surgery is performed by the surgeon After the surgery the patient is moved to the Intensive Care Unit (ICU) where the vital parameters of the patient are monitored. He is then moved to his ward and kept under observation for a few more days and then discharged.

CONCLUSION AND FUTURE WORK

The paper has focused on the essential and extended Object Oriented features of a HealthCare Database 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, HealthCare Database Management System 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 HealthCare Database Management System 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, 2007. The Case for Object Oriented Databases" IEEE Spectrum, 0018-9235/91/ 0002-004, pp: 44-48.
3. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy and William Lorensen, 0000. 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.
6. Dionisis X. Adamopoulos and George Pavlou, 2007. University of Surrey, England Constantine A. Papandreou, Hellenic Telecommunications Organization (OTE), Advanced Service Creation Using Distributed Object Technology, IEEE Communications Magazine 0163-6804/02,
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
25. Balamurugan, S. and P. Ganesh Kumar, 2008. Design and Analysis of Object Oriented Software using UML in different perspectives, International Conference on Advanced Computing Technologies (ICACT 2008), Sponsored by TATA CONSULTANCY SERVICES TCS, Gokaraju Rangaraju Institute Of Engineering and Technology, Bachupally, Kukatpally,Hyderabad-500072 andhra Pradesh, India.