

Application of UML in Real-Time Embedded Systems

Aman Kaur

King's College London, London, UK

Email: aman.kaur@kcl.ac.uk

Rajeev Arora

Mechanical Engineering Department, Invertis University, Invertis Village, Bareilly.

E-mail: rarajeevmanik@gmail.com

Abstract-

The UML was designed as a graphical notation for use with object-oriented systems and applications. Because of its popularity, now it is emerging in the field of embedded systems design as a modeling language. The UML notation is useful in capturing the requirements, documenting the structure, decomposing into objects and defining relationships between objects. It is a notational language that is very useful in modelling the real-time embedded systems. This paper presents the requirements and analysis modelling of a real-time embedded system related to a control system application for platform stabilization using COMET method of design with UML notation. These applications involve designing of electromechanical systems that are controlled by multi-processors.

Keywords:

UML, embedded systems, platform stabilization

1. INTRODUCTION

In today's world real-time embedded systems are everywhere. Often more or less unnoticed they fulfill the task to control the behavior of technical systems in our homes, cars, offices, in the hospitals and at various other places. In all these cases the complexity and number of functions realized in software is increasing rapidly. Often these systems are still developed using older software engineering technologies. As a consequence, it does not fulfill our expectations concerning the needed quality for the real-time systems expressed in terms of reliability, security, timeliness, maintainability, reusability etc. Furthermore, it often suffers from a separation of functions and data as well as from the lack of well-defined subsystem boundaries with precisely documented interfaces.

Because of the above, object-oriented and component-based methods are now a day's more or less accepted means for the development of embedded real-time system software. These methods

promise to facilitate the development, deployment, and reuse of software components with well-defined interfaces. This line of development started at the end of the last millennium with the announcement of the first generation of CASE tools that were able to generate code for embedded system targets from high-level models e.g. defined in the Unified Modeling Language (UML).

UML is very useful in modelling the real-time embedded systems. There are thirteen diagram types to describe various structural, behavioural and physical aspects of a system. The UML gives only a set of notations and not a method. Certain notations in the UML have particular importance for modelling embedded systems, like control system applications [1].

The platform stabilization is one of the typical control applications. These systems are multi-disciplinary in nature which revolves around optical, electro-optical, mechanical and electronic components. The basic functionality of these systems is to provide the user with a stable vision output of the optical or electro-optical sensor in spite of the angular disturbances experienced by the vehicle on which these sensors are mounted. To achieve this functionality, embedded systems are developed which get the inputs from various sensors mounted on the platform, perform control algorithms on these inputs and drive the motors from the corresponding outputs to stabilize the platform against the vehicular disturbances. These embedded systems involve the design of hardware as well as software components.

2. RELATED WORK

The traditional development process for embedded systems is usually cyclic. The engineer often prototypes an algorithm, tests it on a specific hardware architecture, and then refines the software to make most efficient use of the underlying hardware. If software is to be embedded, this type of development can be very expensive and time consuming as performance analysis is done after system components have been functionally tested and integrated.

These complexities of the embedded systems are increasing with the demand of more functionality to be introduced in the systems with limited resources and short time-to-market concepts at lower costs. These issues in turn can be handled by using the reusable components in the systems which can increase the productivity and as well as the reliability. Already the model-driven and component based approaches are being used in the development of real-time embedded systems as compared to the traditional sequential approaches.

A system is represented in UML using multiple models through the 13 diagrams types. Each model describes the system from a distinctly different perspective. According to [2], at the top level the following three kinds of views of a system are defined:

Structural Classification: In this view, the various objects or components and their relationships with each other are considered. It represents the static view of the system through class diagrams, use case view through use case diagrams, the implementation view through component and deployment diagrams.

Dynamic Behaviour: It describes mainly the application's dynamic behaviour i.e. the behaviour over time. This view includes the state machine view through state machine diagrams, the activity view through activity diagrams and the interaction view through sequence or collaboration diagrams.

Model Management: This view describes the organization of the system models into hierarchical units. It uses the class diagrams to represent the organization. The model management view uses the package construct to cross the other views and organises the application's models during development and configuration control.

Two types of views of the system i.e. static and behavioural one have also been described [3]. The static view of the system represents the entities in a system and the relationships among them. The class diagrams, object diagrams, deployment diagrams and component diagrams are used to construct the static model of a system. Behavioural views show how a system behaves over time by showing the order of happening of things, the conditions under which these happen and the interactions between them. The statecharts, use cases, sequence diagrams, collaboration diagrams and activity diagrams are used to construct the behavioural model of the system.

Because of the increasing complexity of the systems, there are two main risks dominating in the real-time systems design. The first one is the risk of functionality that is caused by incompleteness or misunderstanding the requirements from the user. The second one is the risk of performance. It may arise from insufficient performance evaluation in the early stages of the design itself. As a consequence, it does not fulfill our expectations concerning the needed functionality and performance. These both things if uncovered or realized after the development of the system may cause heavy costs to come back to the design of the system with required specifications.

To overcome these problems, object-oriented and component-based methods are now a day's more or less accepted means for the development of embedded real-time system software. They facilitate the development, deployment, and reuse of software components with well-defined interfaces [4]. Thus, By adopting the UML notation, development teams can communicate among themselves and with others using a defined standard [1], [5], [6]. There are already some related work reported in [7], [8], [9], [10] on application of UML to embedded systems design.

In this paper, we present the application of UML notation using COMET method into developing the platform stabilization control system. In particular, we focused on requirements and analysis modelling of the system. COMET [5] is a method for designing real-time and distributed applications that integrates object-oriented and concurrent processing concepts and uses the UML notation [11],[12].

3. CONTROL SYSTEM APPLICATION

A. Introduction

The platform stabilization is basically to reduce the jitter in the sighting systems introduced by the vehicular motions. This is achieved by providing isolation between the sighting system and the mounting platform. The platform stabilization techniques started with mechanical stabilizers. These were the solutions without use of electronic systems. Now a days, electro-optical sighting systems are used as payloads and electronic means are used to control the mounting platforms to stabilize the same against vehicular disturbances. The systems contains various electro-optical sensors for vision, gimbal mechanical assemblies, servo sensors for getting information about the position of gimbals and rate of disturbances experienced by the mechanical assemblies on which the EO sensors are mounted and embedded electronics to implement the signal conditioning of

signals, control algorithms, processing of controller outputs to drive the actuators & providing various sub-system interfaces. These systems needs to be considered and developed using a well-defined control software architecture. In addition to this, it also needs to develop and integrate the platform stabilization system components in a systematic and comprehensive way as it involves the interaction of the system with humans, the software and hardware components having many-to-many relationships, the dependence of performance of the system on the architecture and reuse of the components etc.

These systems interact with humans for getting the commands from them to operate the system in various modes, displaying the gathered imagery data from the optical or electro-optical sensors, informing the users about the health status of the system etc. Thus, well-defined control architecture is required for coherently and systematically combining these services into an integrated system. The block-level representation of the electronics architecture of the system is shown in Figure 1.

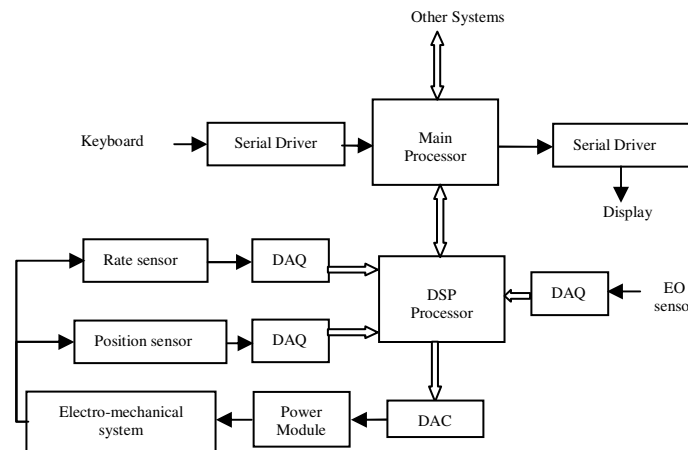


Figure 1: Block diagram representation of the system.

B. Functioning of the System

The system can operate in various modes. The different servo sensors are used as input to the control algorithm for different modes. The control algorithms are also different for different modes. The mode selection is done by the user through keyboard. The keyboard reading is read through serial channel and saved in the memory. On power-on, the system checks the important system parameters, if everything okay, it displays ready status on the display. If some error is there, it will be displayed on the display unit. In case of serious errors, the system does not switch over to the operational mode selected by the user that means the actuators driving the electro-mechanical system will be off.

On selecting the mode, the user needs to feed the required parameters related to the specific mode. For example, if operating in position mode, it needs to feed the position data in x and y direction so that the EO sensor can be positioned accordingly.

During the normal operation, the user can request the system to display the health monitoring data to get information about various system parameters. It can be done by giving the command from the keyboard. The system will acquire all the parameters and pass to the display unit to display it to the user.

In addition to this, the system is also interfaced with other units. This requires continuous exchange between the systems. This does not need a command from the user. Rather, the system itself checks if another unit is available and carry out the transfer of the data among the same.

4. UML MODELLING OF THE SYSTEM

As mentioned earlier, the COMET method has been used to model the system. The requirements modelling and analysis modelling of the above system has been carried out using VP-UML and has been represented in the following paragraphs.

A. Requirements Modelling

The first phase in development of an embedded system either hardware or software is functional requirements capturing. This basically defines what the system is supposed to do. In case of UML, the requirements are captured using use-case diagrams from external actors' point of view. A use-case tells what the system does for each actor. In the present system (Stabilization System), the actor can be a human user as well as the external I/O devices. The use-case diagram for stabilization system is shown in Figure 2. The use-cases shown in the frame stab_control_system are the functions identified that the system is supposed to do.

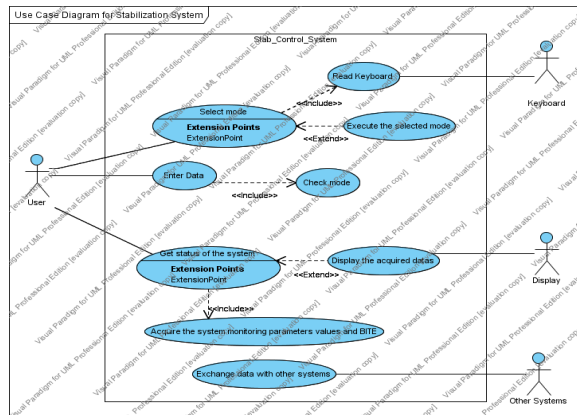


Figure 2: Use Case diagram for stabilization system.

The use-cases are generally accompanied by their description in terms of brief description, pre-conditions for that use-case to happen, post-conditions showing the result of the use-case and flow of events within the use-case. These are given in the following tables from Table I-III.

TABLE I
USE CASE (SELECT MODE) DESCRIPTION

| | | |
|--------------------------|---|---|
| Brief Description | This use case is used to select the mode of the system in which it will be operating. It can select either of the calibration, position or stab mode. | |
| Preconditions | The system should be switched on. It will pass all the health monitoring parameters and ready to switch to any mode. | |
| Post-conditions | The motors will be switched on to operate in the selected mode. The selected mode and the related parameters will be displayed on the display device. | |
| Flow of Events | Actor Input | System Response |
| | 1 | Switch on the system. |
| | 2 | All the devices in the system will be powered up except the motors. |
| | 3 | Select health monitoring data display. |
| | 4 | The system will check all the devices and display the status. |
| | 5 | If everything ok, select mode from the keyboard. |
| 6 | The system will switch over to the selected mode and drive the motors. | |

TABLE II
USE CASE (GET STATUS OF THE SYSTEM) DESCRIPTION

| | | |
|--------------------------|--|---|
| Brief Description | This use-case is required to acquire the current status of main parameters and the main devices. This is to check the health status of the system and updated values of system parameters. | |
| Preconditions | System should be power-on. | |
| Post-conditions | The updated parameters are acquired and displayed on the display unit. In case of any parameters out of range, the indication is given on the display. | |
| Flow of Events | Actor Input | System Response |
| | 1 | Give the command from the keyboard to update the values of all selected parameters. |
| | 2 | The system collects the updated values of the selected parameters and display. |

TABLE III
USE CASE (EXCHANGE DATA WITH OTHER SYSTEMS) DESCRIPTION

| | | |
|--------------------------|---|--|
| Brief Description | This use-case transfers data between the stab-system and other system units. | |
| Preconditions | System should be power-on. It should be either in position or stab mode. All the mode related parameters should be ok. | |
| Post-conditions | Depending on the mode, the system will transmit/receive the data. This data is used in the system for the particular mode selected. | |
| | Actor Input | System Response |
| Flow of Events | 1 | Select the mode |
| | 2 | Execute the mode |
| | 3 | Check the connectivity with other systems |
| | 4 | If everything ok, exchange the data |
| | 5 | check the mode and parameters values on the display. |

B. Analysis Modelling

Static Modelling

The real-time embedded systems continuously interact with the environment and respond to the events. Therefore, it is very much essential to understand the interface between the system and the external environment. The static modelling is done for showing these interfaces and to describe the static structure of the system by developing a system context diagram. A system context diagram provides a more detailed view of the system boundary than a use-case especially for real-time systems. The system context class diagram for stab_control_system is shown in Figure 3.

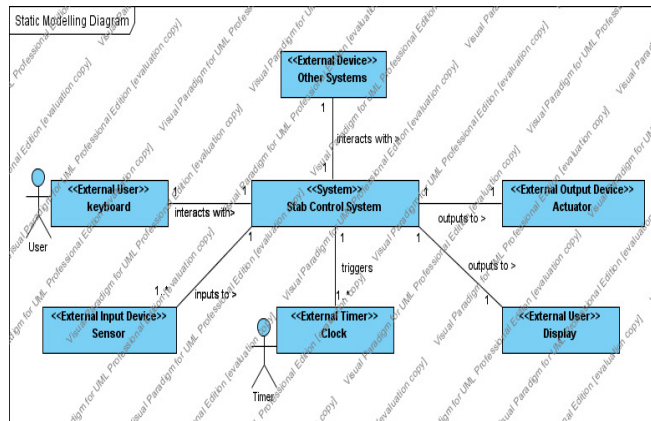


Figure 3: System Context Class Diagram for Stab_Control_System.

In the stab_control_system, the user enters the mode and data through the keyboard. The system uses the sensor data of various sensors like EO sensor, position sensor, rate sensor etc. and it

controls the movement of the mechanical gimbal assembly depending upon the mode and respective data entered by the user. A timer actor is required which manages the timing requirements for the system. These timings are required for processing unit, generating sampling rate for implementing control algorithms, sampling EO sensor data, timings for updating display, reading of keyboard etc. The display unit is required to display the imagery data from EO sensor and the status related information of the system. The system interacts with other systems classes to transfer the interrelated data.

In the present context diagram, the system has been shown as a class with stereotype “System” and the external environment is depicted as external classes to which the system has to interface. The external classes are also categorised using stereotypes like “External User”, ‘External Device”, “External Input Device”, ‘External Output Device”, “External User” and “External Timer” etc.

The next step is to structure the Stab_Control_System into objects. The objective of the object structuring is to decompose the problem into objects within the system. The object structuring diagram for the system has been shown in Figure 4.

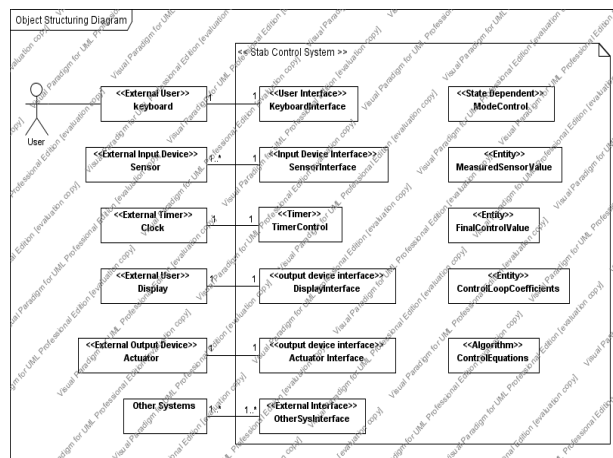


Figure 4: Object Structuring Diagram of the System.

As shown in the Figure 4, the following objects have been identified:

- Interface objects: Keyboard Interface, Sensor Interface, display Interface, Actuator Interface and Other system Interface. These are identified by the external classes that interface to the system.
- Entity objects: Measured Sensor Value, Final Control Value, Control Loop Co-efficients. These objects are required to store the information.
- Control objects: Mode Control, Timer Control. These objects are required for the overall coordination of the system.

In addition to the above, “algorithm” objects are also identified which are responsible for the control algorithms to operate the system in various different modes.

These identified objects will be considered in the preparation for the dynamic modeling of the system.

Dynamic Modelling

The dynamic modelling is of significance importance in real-time systems design. This emphasises on the dynamic behaviour of the system. In the previous section, the use-cases were refined to show the interaction among the objects. To model the dynamic behaviour of the system, UML uses the sequence and collaboration diagrams. These two kind of dynamic diagrams fall in the category of Interaction diagrams. These diagrams are used to describe the flow of messages between the objects. Whereas, the sequence diagrams emphasises on the order in which the messages or events are occurring, the collaboration diagrams focus on the relationships among the objects. Hence, the sequence diagrams are useful in describing the procedural flow through many objects and collaboration diagrams are useful for visualizing the way several objects collaborate to get a job done.

In addition to this, the state dependent objects can also be further explored using finite state machine diagrams.

In the present system modelling, the collaboration diagrams have been developed showing the sequence of events occurring in the system through the interactions among the objects. The same has been shown in Figure 5. The rectangles in this figure represent objects not classes. The arrows on the links represent messages. These are labelled with their name and sequence numbers.

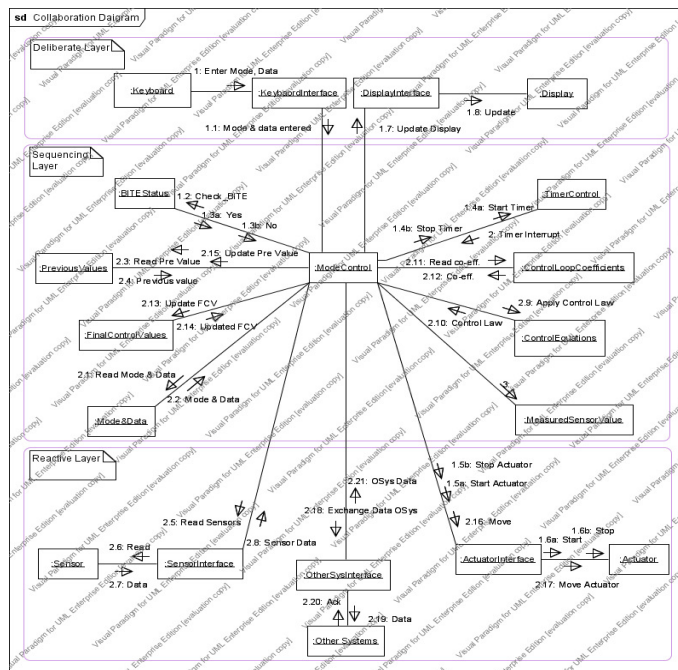


Figure 5: Collaboration diagram.

In the actual system design, the modular approach has been adopted where different sub-units have been integrated based on their functionality. Here, the same has been shown as different layers. The three layers has been chosen are: deliberate, sequencing and reactive layers. The functionality of these layers has been divided as follows:

- Deliberate Layer: Interface with user and execute a planning process.
- Sequencing Layer: Controlling the part that executes the process by managing the components in reactive layer and extracting information from the sensors.
- Reactive Layer: Controls the real-time command and hardware related modules like sensors, actuators and other systems.

In the collaboration diagram, two sequences have been shown. The sequence no. 1 & 2 are further elaborated in the following paragraph.

The sequence no. 1 mainly represents the initialization of the system. The mode and data related to the same are entered by the user through keyboard interface. These are saved in the memory. Health of the system is monitored. If everything ok (i.e. 'Yes'), the system starts the timer for proving interrupts to execute the control laws. The actuators are also switched on. The required information is sent to display for updating the user about status of the system. In case, the BITE check status is not ok (i.e. 'No'), the timer for control algorithms and the actuators are switched off.

The actions of sequence no. 2 start with the timer interrupt. It is important to mention here that the timer used for the control law implementation is periodic as the performance of control law depends upon the sampling rate when implemented in digital domain. When this interrupt occurs, the system first reads the mode set by the user and the respective data. In feedback control loops, the present output depends not only on the present inputs but also on the previous inputs and outputs. Hence, the next step is to read the previously stored values. Depending on the selection of mode, the particular servo sensor value is read. The control law is implemented using these inputs and the corresponding co-efficient values. The output of the control law is the final control value (FCV) which is used to drive the actuators. The previous values are updated before the occurrence of second timer interrupt. The required data is exchanged with the other systems.

The present system Stab_Control_System is having a state dependent object i.e. Mode Control. Therefore, there is need to develop a state chart that will be executed by Mode Control object. The state dependent dynamic analysis addresses the interaction among objects that participate in the state dependent use cases. As per [5], the state chart must match with the collaboration diagram and specifically it is required to take into account the messages that are received and sent by the control object, which executes the state chart. Hence, the events shown on the state chart diagram must be consistent with the input events of Mode Control object. Similarly, the output events must be consistent with each other. Same consideration has been taken while deriving the state chart shown in Figure 6.

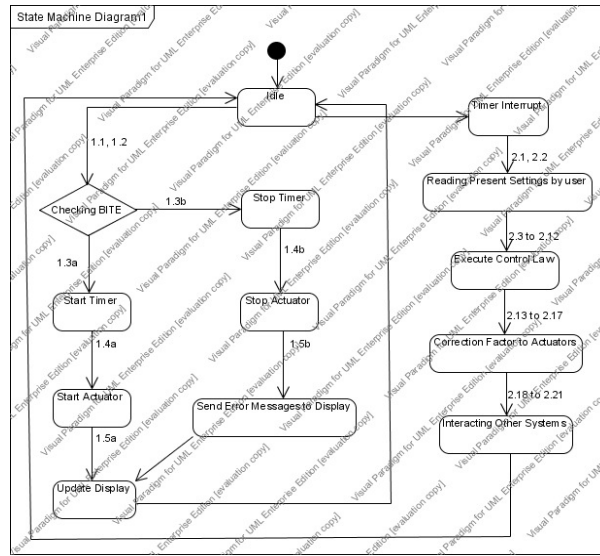


Figure 6: State diagram.

Normally like any embedded system's infinite loop, the system is shown in idle mode and on occurrence of either input by the user or the interrupt generated by the timer triggers the next cycle of operations of the system. In the same way, the outputs of the system i.e. controlling of actuators, display and other system also is consistent with the collaboration diagram.

5. CONCLUSION

This paper has presented the requirements and analysis modelling of a control system using object oriented concept using UML/COMET method. Through these implementations, it has been found that the UML standard is very useful in requirements capturing, decomposing the system into objects and defining their relationships. The control system application models have been developed using three UML modeling mechanisms: use case modeling, object modeling, and dynamic modeling respectively. Through requirements modeling and analysis modeling, the usability of the Use Case, Class/object, Collaboration and State-Chart diagrams has been exploited. The models presented in this work are obtained after lot of iteration going forward and backward between static and dynamic models. The models have been developed irrespective of the target platforms on which finally they will be executed, hence, enabling their reuse over different operating platforms. The approach used in this work shows the possible adoption of UML in the embedded system design .

6. REFERENCES

- [1] G. Martin, L. Lavagno, and J. Louis-Guerin, "Embedded UML: a merger of real-time UML and codesign", CODES 2001, Copenhagen, April 2001, pp.23-28.
- [2] Rumbaugh, J., Jacobson, I. and Booch, G. 1999. The Unified Modeling Language - Reference Manual. *Reading, MA: Addison Wesley Longman.*
- [3] Holt. J. 2001. UML for Systems Engineering. *United Kingdom: The Institution of Electrical Engineers.*
- [4] Hofmann, P. P. and Schurr, A., 2002. OMER – Object-oriented Modeling of Embedded Real-Time Systems. *Post-proceedings of OMER-1: May 28/29 1999, Herrsching am Ammersee and OMER-2: May 10-12 2001, Herrsching am Ammersee.*
- [5] H.Gomma, Designing Concurrent, Distributed, and Real-Time Application with UML, Addison-Wesley, 2000.
- [6] G. Martin, "UML for Embedded Systems Specification and Design: Motivation and Overview", DATE 2002, March, 2002.
- [7] E. Riccobene, P. Scandurra, A. Rosti and S. Bocchio, "A SoC Design Methodology Involving a UML 2.0 Profile for SystemC", Proc. Design, Automation and Test in Europe, 2005.
- [8] T. Schattkowsky and W. Muller, "Model-Based Design of Embedded Systems", Proc. 7th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing, 2004.
- [9] N. A. Zakaria, M. Kimura, N. Matsumoto, N. Yoshida, "Stepwise Refinement in Executable-UML for Embedded System Design: A Preliminary Study", World Academy of Science, Engineering and Technology 55, 2009
- [10] G. Wang, "Modeling C-based Embedded System using UML Design" Proceedings of IEEE International Conference on Mechatronics and Automation, 2009
- [11] OMG Unified Modeling Language, Version 1.5, March 2003. Available at:<http://www.uml.org>
- [12] M. Fowler and K. Scott, UML Distilled 2nd Edition, Addison Wesley, 2000