A Comparative Study of Software Engineering Techniques for Real Time Systems

Mrigank Shekhar
B. Tech Computer Science and
VIT University
Vellore

Mayank Shekhar
B. Tech Computer Science
VIT University
Vellore

Ayush Gupta
B. Tech Computer Science
VIT University
Vellore

ABSTRACT
Designing and developing software for Real-time is a challenging task. Issues related to real-time control and embedded system are involved in the software development process. This type of software must be developer with proper software methodology or well-defined development process in order to increase the productivity and quality of the software design and software products. This paper would examine and compare four of the most common methodologies used in real time software development. The methods selected for comparison are CORE, ROOM MASCOT and UML. The methods are compared among themselves based upon attributes such as usability, compositionality and proper RT (real time) notations available. The paper discusses in detail the various notations available in every methodology and ranks them based on their merits and de-merits. The paper aims to reach a logical conclusion over the use of which real time methodology results in most apt software development

General Terms
Real-time systems, Real-time software Methodology, Comparative Study

Keywords
Real-time systems, Embedded systems, Real-time software Methodology, CORE, ROOM, MASCOT, UML

1. INTRODUCTION
Development of real time systems i.e their analysis and development is an intricate process. It is so critical due to the fact that Real-time systems have different properties when compared to other systems as they have timing constraints. These systems are usually put under environment where timing and scheduling are of the utmost importance. The type of software must be developed with proper software methodology or well defined software process. Object-orientation is a powerful approach to managing complexity, which has received widespread attention in the last years [14]. These systems need special timing communication and reliability requirements which are not easily explained by the usual methodologies. Minor changes to specification could be very costly as real time (Embedded systems) function as a whole and any change to one phase has a severe effect on others. The software development for these systems is harder because they are embedded directly into the hardware and frequent changes may require entire re-writing of the system. Problems occur when software engineers do not have the correct understanding of the processors, devices and the operating systems used.

Various methods and notations have been developed for analysis and designing of real time systems. They differ from the normal methods as these focus heavily on event driven behavior, communication and timing issues while also concentrating on the usual system properties. CORE and MASCOT have a data driven approach i.e they focus on traditional structured analysis and design whereas ROOM and UML are relatively newer methodologies which use object oriented notations [7].

2. PROBLEM STATEMENT
Due to the availability of a wide variety of methodologies there is no standardization as to which methodology yields the best software engineering techniques. All of the available methodologies focus on one particular aspect of software engineering. This leads to inconsistency among the quality of various phases of the process. Usage of standards, procedures and consistent methods should be followed throughout various phases to produce the highest quality software. Problems appear in a way as

- CORE focuses more on the needs of the requirement rather than providing a complete solution.
- ROOM involves a CASE tool called ObjectTime which limit many of design choices like multiple inheritance.
- MASCOT relies heavily on directly building a model and has very less support for requirement analysis.
- UML due to its simplified and informal nature is not good for managing a software development process.

3. PROPOSED SOLUTION
The paper will look to provide a feasible solution as to which methodology would provide an optimal software engineering process. It will compare the aforementioned methods based on usability, compositionality and the real time notations available and give an analysis of which method is better from the other and on which basis. The paper should give an indication over how a methodology fairs against the other.

4. EXPLANATION OF METHODS
4.1 Controlled Requirements Expression
CORE is one of the many software engineering techniques available with heavy focus on the gathering of requirements and how are they formulated. Originated for use in avionic industries the requirement analysis phase is of prime importance and the project is divided into subtasks based on viewpoints. I) Problem and requirement definition, II) gathering of data, III) development of detailed models for each viewpoint and v) combination of single viewpoints into a composite one. CORE defines the steps in the

production of a requirement specification with particular emphasis on startup and link between steps. It can be used in conjunction with object oriented analysis and mainly works for gathering of requirements with informal block diagrams [3]. CORE is basically a systematic expression of the requirements that are needed for real time analysis and design. The focus is mainly on requirements rather than design.
The main limitations of CORE are that: i) timing, concurrency and synchronization issues are not properly explained and therefore forgoes the whole aim for usage with critical real-time systems. ii) it is unsuitable for architectural design iii) it is rigidly focused on several steps [8].

4.2 Real Time Object Oriented Modeling

ROOM is another methodology followed in real time system development which lays heavy focus on development of the project by concentrating more towards the physical design of the project. ROOM at it's core utilizes simple state charts much like UML and also uses ROOM chart diagram based on the 'actor' concept. The state charts help in the final formulation of the code as the state chart itself is a graphical representation of different parts of the code and how the control flows through them. ROOM has very limited resources to formulate the initial requirement phase and is also very limited because its implementation need the use of a particular CASE tool called 'ObjectTime' [6]. The actor initiates a sequence of events. Ports are used for communication, threads control behavior [7]. Limitations of ROOM are: i) tied with one particular CASE tool called 'ObjectTime' ii) a limited number of diagrams showing only certain views are available [8].

4.3 Modular Approach to Software Construction Operation and Test

Mainly used for avionics and in the military field, MASCOT is a highly modular rigorous approach based on hierarchical decomposition to lower levels. Based on processes and activities, it strives to achieve highly interactive real-time systems in a structured way. It heavily focuses on communication between different components and needs the specification to be fully completed at every level. Highly detailed interfacing between modules helps maintaining concurrency and synchronization. The main steps include i) describing the overall internal functions and external connections of the system into a network ii) The network is decomposed into lower-level components, iii) Components are coded in terms of algorithms and data structures [5]. Rules which are followed include the limitation of direct data communication between processes, communication occurrence only through specified channels and well defined info exchange areas must be used for exchange, storage and communication of data.

Its limitations include-i) No direct support for requirement analysis and the process is directly followed into model creation. ii) Not suitable for prototyping or rapid development cycles. iii) Very expensive to apply [9].

4.4 Unified Modeling Language

UML as the name suggests is a unification of all the popular and effective software engineering techniques available like ROOM, CODARTS etc. UML state diagrams are simplified STDs, communication diagrams are found elsewhere as interaction diagrams, sequence diagrams are derived from Message sequence charts. It builds on the functionalities lacking in other methodologies and improves them in the fields on which they lacked by standardizing them. It has a wide support with a variety of CASE tools and can be implemented without formal knowledge. Two main system views can be categorized into i) static ii) behavioral. UML, as is, is not a proper software development tool and is usually combined with other software tools [7].

5. METHOD COMPARISION

The methods have been compared using four fundamental issues these are i) usability, ii) compositionality, iii) Proper RT notations available and iv) ranking score.

5.1 Usability

Usability explains the ease of use of the method, CASE tool support. This is important because methods that are easy to use are preferred to those that are more complex. Certain notations are better to describe activities. Other notations are more suitable for explaining communication between components [7].

5.2 Compositionality Score

Compositionality describes how the notations in the method fit together. It also describes the overall structural composition. This is important because this structure will be used to construct the final system. This is based on the number of diagrams / notations used. The more notations there are the more difficult it becomes to keep consistency. There is recursion which in this case implies the existence of techniques to refine the final design, this involves abstraction and information hiding. There is the possibility to obtain full specifications from recursion or decomposition. This is known as graphical to textual conversion. There is the issue of cross-references between notations. Poor cross-references could imply a problem. Good cross-references imply good consistency between the method's notations [7].

5.3 Proper RT Notations Availability

Proper real time notations imply how well a method describes issues like concurrency, synchronization, event handling and message communication. Real time systems depend on triggers and communication issues. Support for communication constructs includes support for concurrency, synchronization, mutual exclusion, signaling, communication control, ports and abstraction [7].

6. RANKING SCORE

These methods have been compared using detailed observations and experience. To compare the methods a score from 1 to 4 was given for the relevant attribute. The score is as follows 1-poor, 2-average, 3-good, 4-very good and 5-excellent

6.1 CORE

CORE has compared to others moderate ease of access as breaking of the project into various viewpoints takes formal approach. The diagrams and notations facilitated by CORE are very clear and easy to understand as the project is properly broken into layered structure with well-defined boundaries. The notations are not at all confusing. As it is widely used in avionics industry thus it has good CaseTool support also. Diagram consistency, due to the nature of the methodology of only supporting the requirement phase, is pretty high as there are not so many notations that the consistency is lost due to muddle up. Graphical to textual conversion is very bad as we know that core supports only the requirement phase properly and therefore the conversion of the graphical methods to full specification is not properly achieved [7]. Cross reference is

pretty good in this methodology as the notations are closely inter related and cross referable as they are only working on one part of documentation usually. Core, when coming to notation support, is very bad. With all the specialization it has over the requirement phase, it does very poorly in supporting RT notations. Message Comm., Resource Management ,Timing Requirement notations are very bad and have almost no support. Although special event type does have decent backing.

6.2 ROOM

ROOM although has simple state charts as its basic units of diagrammatic representation, still the usage of actors concept makes it very tricky to implement and therefore has very low ease of use. Clarity of diagrammatic notations is fairly good as the diagrams clearly depict how the flow of control occurs in the code. Tool support, although flexible due to direct implementation through C++ code, is still very limited as 'ObjectTime' is the only main tool available. Diagram consistency, due to the nature of the methodology and its constraint in the OO domain, is pretty high as there are not so many notations that the consistency is lost as state charts are the only dominant notations. Graphical to textual conversion is exceptionally good as ROOM specializes with its tool in the conversion of graphical notation to implementable code and therefore the conversion of the graphical methods to full specification is properly achieved [7]. Cross reference is decent in this methodology as the notations are only working for the state charts and the actors so as to facilitate easy referencing. ROOM, when coming to notation support, has good support. Message Communication has enough support to be implemented without hassle while, Resource Management and Timing Requirement notations have individual and well detailed notations which give ROOM an edge over others in RT system implementation. Although special event type do not have downright individual notations, still the support exists to a satisfactory level.

6.3 MASCOT

Ease of use under MASCOT is very bad as the requirements are very strict for proper implementation of the project so as to keep the real time requirements like timing and sync under control. The specification at every level needs to be completed so as to proceed to another level leading to very steep learning curve. For similar reasons, clarity is not achieved in notations as the diagrams are very complex layered structure. Also tool support is very scanty as very few tools actually support the methodology. Diagram consistency, as due to various restrictions applied at each level of the designing , is pretty high as there are no conflicts or lack of information at any level to have inconsistency and all the connections at every level, whether internal or external to a module, are well implemented. Graphical to textual conversion is very good as MASCOT has in its stages the production of a network diagram which links all the modules and there functions and these can be decomposed to lower level components which can be coded in terms of algorithms and data structures and therefore the conversion of the graphical methods to full specification is properly achieved [7]. Cross reference is decent in this methodology as the notations are closely bound together by a set of specifications which always need to be fulfilled and thereby cross referencing is very decent. Message Communication due to the closely wound structure is very good and the notation availability is pretty high. Also resource management is also very high as the specifications are clearly known and need to be compulsorily fulfilled in order to proceed and thereby managing the resources properly. Timing requirements although having good support are not supported to the extent as others. Special event support is very high as all of these are carefully covered through the different specification levels.

6.4 UML

UML provides a large variety of informal notations and diagrams which gives it an advantage over other methodologies used for Real time software development. No training is required to develop software using UML because of its clear and simple nature. It provides Object-oriented way of software development with no strict formal structure or terminology. UML has brilliant tool support due to wide use in the industry across various disciplines. Diagram consistency, due to too many notations and diagrams is very low because keeping track of each and every notations and diagrams in a large project diagrams is a difficult task and requires effort. UML lacks support for graphical to textual conversion since UML has a graphic notation for each and process and activity and representing them in a textual way is very difficult and time consuming [7]. Cross reference is pathetic in this methodology due to too many notations used which leads to ambiguity. Message communication, is very nominal due to no standardization. It has almost negligible support for Real time resource management and Timing requirement because Real-time systems are, by definition, constrained by some aspect of time [15]. It is critical to capture this timing information while modeling the system to specify a reasonable system design which is difficult when using UML methodology [15].

Table 1. Usability Score

Method	Ease Of Use	Clarity Of Diagrams/Notations	Case Tools Support
CORE	3	4	4
ROOM	2	3	3
MASCOT	1	2	1
UML	5	4	5

Table 2. Compositionality Score

Method	Diagram Consistency	Support for Graphical to Textual Conversion	Cross References
CORE	3	2	4
ROOM	4	4	3
MASCOT	4	4	4
UML	1	2	1

Table 3. Proper Real-Time Notation Score

Method	Message Communication	Support for RT resource management	Support for Timing Requirements	Support for special event types
CORE	1	1	1	4

ROOM	3	4	4	3
MASCOT	4	4	2	4
UML	2	1	1	1

7. RESULTS

The usability results indicate that UML is the best usable method followed by CORE. The compositionality results indicate that method MASCOT has better compositionality. This is because of good cross-referencing between notations. There is also the fact that this method support proper graphical to textual conversion in detail. This is not so with the UML. The comparison of the real time notations, indicate that some RT methods seem to have better notations and compositionality than others. This is possible because these methods and its notations have been refined over a number of years. These results just give an indication of some attributes. It is possible to derive other combinations if there are certain requirements.

8. CONCLUSION

The final conclusion is that there is no single method that is overall the best method on all attributes. It is evident that the UML and modern methods cannot solve certain issues tackled by data driven methods that were designed precisely for real time(like MASCOT). The UML is a more general language that tries to cover many different types of systems and scenarios at the expense of certain detail. Solutions to this could be to extend UML via stereotypes like allowing the designing to follow a pattern similar to that of MASCOT of forced specification fulfillment at every stage. Another advantage of UML is that some UML diagrams are applied in a MDA approach. The UML has the advantage of gaining widespread use and a lot of work is being done to improve UML continuously. UML does not have proper control flow diagrams similar to those found in ROOM or MASCOT. These are important for designing command and control and embedded system tasks. UML instead uses activity diagrams or communication diagrams. Activity diagrams are more adequate for business analysis, communication diagrams lack some detail and need modification on the other hand control flow diagrams are oriented to task management, reactive behavior and control. This could indicate that UML is more oriented towards building soft- real time systems like those used in e-commerce, agent architectures, workflow systems, etc. The UML has given the initiative to create other modeling concepts and methods like AGILE and FMCs(Fundamental modeling concepts). Methods like MASCOT and ROOM have been directly designed for hard real time systems like avionics, cruise control, etc. These are quite rigorously demanding and require the use of specific constructs and possibly even languages. MASCOT, ROOM and UML-RT whilst being suitable for describing complex RT systems, unfortunately lack widespread support of many CASE tools and require time to master. A practical approach is suggested. It does not make sense to restrict use to a single method. This is that when using one particular method one should possibly also consider using notations from another method as is required by the nature of the problem. Depending on the nature of the system being modeled a method should be selected. Some methods are more suitable for business workflow systems. Others are more suitable for hard event-driven real time systems like those used in avionics and control systems. Therefore the optimal software engineering methodology for RT systems will only be achieved by grouping together the functionality of UML with the hard real time constraints of MASCOT.

9. REFERENCES

[1] Formal methods and notations applicable to telecommunications, IEEE Tutorial colloquium, pp 5/1-5/4.

[2] Applicable Modeling, Verification and analysis for real-time systems, p 2/1-2/4.

[3] CORE - A method for controlled requirement specification g.p.mullery.

[4] www.drdobbs.com/real-time-object-oriented modeling/184410342.

[5] MASCOT as a design tool (software engineering education) , IEEE Colloquium, 6/1 - 6/4.

[6] Automatic Implementation of Real-Time Object-Oriented Models and Schedulability IssuesSaehwa Kim, Sukjae Cho, and Seongsoo Hong.

[7] A Comparison of Software Analysis and Design Methods for Real Time Systems Anthony Spiteri Staines International Journal of Computer, Information Science and Engineering Vol:1 No:3, 2007

[8] An Efficient Object-Oriented Variation of the Statecharts Formalism for Distributed Real-Time Systems by Bran Selic.

[9] Mascot 3: an informal introductory tutorial, IEEE Tutorial colloquium.

[10] Real-Time Systems : Design Principles for Distributed Embedded Applications By Hermann Kopets 1997.

[11] Experimentation in Software Engineering By Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén.

[12] 14. A Lightweight, Component-Based Approach to Engineering Reconfigurable Embedded Real-Time Control Software by Jagun Kwon ; Univ. Coll. London, London, UK ; Hailes, S.

[13] Embedded systems in real time applications, design & architecture August 2005 by A. L. Suseela, V. Lalith Kumar.

[14] REAL-TIME OBJECT-ORIENTED DESIGNAND FORMAL METHODS Juan Antonio de la Puente.

[15] Werner Van Belle, Tom Toutenel, Viviane Jonckers; Real Time UML; SEESCOA Deliverable d2.1; 26 pages; April,2000.