A Brief Comparison of Real-Time Software Design Methods

Tony Spiteri Staines
Dept. of Computer Information
Systems
University of Malta
staines@cis.um.edu.mt

ABSTRACT

This paper briefly attempts to compare several mainstream methods/methodologies that are used for the analysis and design of real time systems. These are i) CORE, ii) YSM, iii) MASCOT, iv) CODARTS, v) HOOD, vi) ROOM, vii) UML, viii) UML-RT. Methods i-iii are use a data driven approach, whilst methods iv-vii use an object-oriented approach. All these methods have their advantages and disadvantages. Thus it is difficult to decide which method is best suited to a particular real-time design situation. Some methods like YSM, MASCOT and CODARTS are more oriented towards designing event driven systems and reactive behavior. Object oriented methods like the UML have many diagrams obtained from other methods. In the first part of the paper each method is briefly presented and its main features are explained. In the second part a score based ranking is used to try to identify which method has the best overall characteristics for real time development. The final results are presented in a tabular form and using a bar chart. In addition to this it is explained how each method fits in the SDLC. Both the score of each method and how it fits in the SDLC must be considered when selecting methods. To conclude some other issues are explained, because the selection of one method does not automatically imply that there will not be any problems.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/ Specifications – *Elicitation methods, Languages, Methodologies, Tools.*

General Terms

Design, Measurement

Keywords

Real Time, software design methods, methodologies, systems modeling, comparison of methods, data driven approach, object-oriented approach, software development lifecycle, RT – real time

1. INTRODUCTION

Real Time systems are more complex than traditional systems. Some real time systems demonstrate reactive behavior. They have timing, communication and reliability requirements that are not easily accounted for [10]. RT development involves not only software engineering but also hardware engineering, control engineering and communication engineering. Minor changes to specifications could be very costly. Software programs embedded directly into hardware controller and devices might require entire rewriting. There are hardware configuration problems when software engineers do not understand why specific processors, devices and operating systems are needed. Specific methodologies have been developed for helping the analysis and design of RT systems. These were different from normal methods because they had to focus on behavior and not just static system properties. Methodologies do not guarantee solving all software development problems although they attempt to structure the analysis & development of RT systems applying design techniques and rules. RT methodologies make use of the basic concepts from structured analysis and design [3,4]. First methodologies like MASCOT, JSD, DARTS were data-driven based on principles from traditional structured analysis and design [4,7]. Later work was to use object oriented notations. These offer several advantages over the former methodologies like reuse and a neater approach [2,9]. The OMG boosted the use of object oriented methods for analysis and design through the UML and UML-RT. Early versions of object oriented modeling lacked the dynamic aspects of behavior and focused mainly on static and structural aspects which were insufficient for real time. Later models tried to combine object oriented notations with state charts, activity diagrams, interaction diagrams or message sequence charts now known as sequence diagrams in the UML.

Some authors compare software design methods with software modeling notations. This is incorrect because a proper design method should encompass the entire software development lifecycle process. This is not the case with the UML where the focus is on modeling a system rather than on managing the entire software development. The OMG created USDP (Unified Software Development Process) based on the UML notations. The USDP is a proper methodology. Even the COMET (Concurrent Object Modeling architectural design method) in [8] combines the UML notations within a special lifecycle model. On the other hand software notations are more generic and focus on particular aspects of the design process. In system specifications there could be the use of models or system views. These could be singular or multiple, formal, semi-formal or informal, graphical or language based [1]. Good specifications using UML constructs could be used to derive specifications in the HDL (hardware description language or ESDL (embedded systems description language) as in [6].

2. OVERVIEW OF SOME METHODS

2.1 Controlled Requirements Expression

The CORE (controlled requirements expression) method [4,5,12] is based on block diagrams, viewpoint diagrams and written statements of requirements. It was designed in the UK for the requirements analysis phase and was widely used in avionics. It is suitable for the informal process of gathering the systems requirements that are expressed using informal notations like block diagrams, viewpoint diagrams etc. It is very simple to use and understand not involving any complex notations etc. This approach is mainly data driven at a very high level but still could be used in conjunction with object oriented analysis. There is top-down decomposition into smaller parts. The CORE makes use of viewpoint diagrams, thread/ dataflow diagrams and block diagram notations. Control loops are available for use in the final diagram. The idea of view points could prove to be important to other methods and also for situations where requirements specification proves to be difficult. The CORE method tries to take a practical approach to problem identification at the earliest possible stage. The diagrams used are quite simple and some form of support could be obtained using modern case tools. The results that are produced using this method can be used as the input for another method. Some limitations are that: i) There is no reference to timing, concurrency, synchronization, ii) Unsuitable for Architectural design iii) No simulation model is produced.

2.2 Yourdon Structured Method

The YSM (Yourdon structured method) in [4] is based on the classic DFDs and structured methods used for traditional data design. It has been adapted and combined with many diagrams for RT design. It has been developed and refined over the years and many modern CASE tools can be used to support the notation. YSM starts off from a high-level and decomposes the system into lower levels ending up with complete program specifications. Two embedded design Methodologies have been derived from YSM. These are Ward-Mellor, Hatley-Pirbhai. This method can being used in conjunction with diagrams like PEM(Processor Environment Model) which is a hardware based design to help decide on the hardware configuration. There is also the SEM (Software-Environment Model). There are many different data driven methods that make use of the principles in YSM and add other diagrams. The PEM model and SEM are important because as pointed out RT systems are highly dependant on the available hardware which is normally ignored. YSM also uses DFDs, STDs, E-R diagrams, textual specifications, structure charts etc for design purposes. DFDs can be combined with STDs to represent both continuous and discrete actions. The behavioral model consists of DFDs, STDs & ERDs together with textual support describing the requirements but having no implementation details. The PEM covers the physical processors deciding which processor should do which work and HCI details. The COM involves translating the SEM units into structure charts and refining them so that this can be translated into program code. One advantage is that YSM is a highly structured data analysis method. Some limitations are i) it is unsuitable for prototyping. ii) it must be followed in logical sequence or sequential ordering for successful implementation iii) It is possible to take a long time to implement the complete system iv) user must have familiarity with certain constructs. v) there is specific reference to timing issues, concurrency etc although the diagrams can be altered to support time.

2.3 Modular Approach to Software Construction Operation and Test

MASCOT (Modular approach to software construction operation and test) was first issued in 1970s by the Royal Signals and Radar Establishment UK and successive versions MASCOT 3 exist [11]. It is mainly used for avionics and in the military field. It is a highly modular rigorous approach based on hierarchical decomposition to lower levels. MASCOT is based on processes or activities in a system and aims at designing complex interactive RT applications in a highly structured approach. Mascot focuses on communication between different components and enforces that a specification must be complete at every level. Interfacing between modules is extremely well represented, thus even concurrency and synchronization can be dealt with. The main steps are i) Describe the overall internal functions of the system, together with its external connections. This is known as the Network Diagram. ii) The network is decomposed into lower-level components iii) Define the structure of single thread processes (transform). iv) Design and code simple components in terms of algorithms and data structures. There are the following rules i) processes cannot send data directly to other processes ii) communication between different components can only take place through channels or windows. iii) Intercommunication Data Areas (IDAs) must be used for data exchange, information storage and communication. Some limitations of Mascot are i) it does not directly support requirements analysis and goes directly into building a model ii) it is not widely supported via many case tools iii) it is not suitable for prototyping or RAD iv) it is expensive to apply.

2.4 Concurrent Design Approach for RT Systems

CODARTS (Concurrent design approach for RT systems) is a modified form of DARTS (Design approach for RT systems) [7]. CODARTS implements concepts from DARTS for an object-oriented perspective. ADARTS was mainly aimed for use with the ADA language. CODARTS uses notations from RTSAD (Real-Time structured analysis and design). The diagrams used in CODARTS are similar to control flow diagrams that use special symbols for different types of message passing e.g. loosely-coupled message communication, tightly-coupled message. Possible diagrams are task architecture diagrams, software architecture diagrams and STDs. CODARTS classifies message passing into several types not normally found in other methods. These are supposed to be easily implemented in ADA. Some limitations of CODARTS are i) Designed mainly for the ADA language. ii) Notations used are not well understood iii) Message communication even though well identified still does not account for concurrency, synchronization, mutual exclusion. iii) uses a limited number of views.

2.5 Hierarchical Object Oriented Design

HOOD (Hierarchical Object Oriented Design) method covers some parts of the software development lifecycle. It is mainly aimed at the ADA language taking an object oriented approach. It can be useful for prototyping. The idea behind HOOD is to identify objects in a parent to child relationship and their operations. Finally a graphical system description is to be produced in a control/ dataflow diagram that shows the flow of information between a set of objects. The diagrams can be decomposed to the required levels. The Top-Level Object is an active object because it uses the lower-level ones but not vice-versa. Rules distinguish passive Objects from active Objects. Certain flows are not permitted like cyclic flows. Limitations of HOOD are : i) does not distinguish Data Flows between Objects from Event Signals ii) Not so simple and straightforward to use iii) Has just one main diagrammatic type thus just one model structure is given.

2.6 Real time Object Oriented Modeling

ROOM (Real time object oriented modeling) is similar to HOOD in principle but is more oriented to RT design and focuses on proper communication between objects. ROOM introduces the concept of Actors in 'ROOMcharts' which are a variation of StateCharts (ROOMcharts define the state of an actor, signals that initiate transitions between states, and actions performed at transitions. There is a strong focus on this actor behavior. The actor is anything that initiates a sequence of events. There is the use of 'ports' for information exchange and threads that can have a priority. Some limitations of ROOM are : i) Closely Tied with one particular CASE tool called 'ObjecTime' which can generate C++ code ii) It has a limited number of diagrams that show only certain views of the system i.e. actor view. iii) Its diagrams need to be supported with temporal analysis for complex systems.

2.7 The Unified Modeling Language

The UML can be considered to be a repository of notations existing in methods like ROOM, HOOD, YSM, MASCOT, etc. The name 'unified' implies a unification of modeling constructs. E.g. UML state diagrams are simplified STDs, communication diagrams are found elsewhere as interaction diagrams, sequence diagrams are derived from MSC (Message sequence charts). It contains notations that are lacking in other methodologies and tries to standardize them and it is set to improve upon previous notations. It is well supported by a variety of CASE tools when compared to other methods and can be used by anyone without formal knowledge. The main system views can be categorized into i) static ii) behavioral. The UML is not a proper software development method and can be combined with almost any development method. Diagrams and notations used are Informal. It is possible to use the OCL (Object Constraint Language) to formalize the diagrams used. When a class uses operations by a second class a control flow is set up. The UML does not distinguish between the spatial distribution of objects and the logical object distribution. Code generation can be done from some UML diagrams like a class diagram. There are projects like the ECLIPSE open source tool that supports many UML constructs. There is a lack of standardization amongst the UML CASE tools and UML versions giving rise to confusion about which notations should be used. Some CASE tools providers have created their own notations that differ from those in the UML. Some limitations of the UML are : i) studies show that maintaining UML diagrams can become a complex process ii) UML lacks formal verification iii) the same thing can be modeled in several different ways, all could be correct. So there is a lack of consistency.

2.8 The UML-RT

UML-RT is based on extensions to the UML specifically aimed at RT. The most important 'new' notations are mainly capsules, ports, connectors and protocols. UML-RT implements some ideas from HOOD, ROOM and MASCOT adding them to the normal UML notations. E.g. the idea of capsule diagrams embedding child objects is similar to HOOD Parent-Child object relationships. The idea of active and passive ports already exists in ROOM. The idea of using capsules to model complex objects that are usually spatially distributed is similar to that of MASCOT where components / devices are connected using windows, ports and IDAs. Some limitations of UML-RT are i) not widely used and supported. UML-RT includes all the modeling capabilities of ROOM.

3. PRACTICAL ASSESSMENT OF THE METHODS

These methods were measured on the attributes in table 1.and 2. The final classification results are in table 3. i) Consistency between notations refers to the consistency between the diagrams used. The more notations there are the more difficult it becomes to keep consistency. ii) Support for communication constructs includes support for concurrency, synchronization, mutual exclusion, signaling, communication control, the use of ports and abstraction. iii) Support for resource control refers to the handling of different system components with processing loops and activity management, possibly used for performance management. iv) Support for temporal requirements indicates the need to show the different states the system or components can be in. Other issues like CASE tool support, abstraction and also ease of use were also considered.

Table 1. Method Comparison 1

METHOD	CONSISTENCY BETWEEN NOTATIONS	SUPPORT FOR COMUNICATION CONSTRUCTS	SUPPORT FOR RESOURCE CONTROL	DIFFERENT SYSTEM VIEWS
CORE	**Very Good**	**Poor**	**Poor**	**Average**
YSM	**Very Good**	**Poor**	**Poor**	**Average**
MASCOT	**Very Good**	**Excellent**	**Very Good**	**Poor**
CODARTS	**Very Good**	**Good**	**Good**	**Average**
HOOD	**Very Good**	**Average**	**Good**	**Poor**
ROOM	**Very Good**	**Good**	**Very Good**	**Poor**
UML	**Poor**	**Average**	**Average**	**Very Good**
UML-RT	**Good**	**Good**	**Good**	**Good**

score method (poor = 1, average = 2 , good = 3, very good =4, excellent = 5

Table 2. Method Comparison 2

METHOD	CASE TOOL SUPPORT	ABSTRACTION / INFO. HIDING & COMPOSITION	SUPPORT TEMPORAL REQUIREMENTS	EASE OF USE
CORE	**Very Good**	**Average**	**Poor**	**Good**
YSM	**Very Good**	**Average**	**Poor**	**Good**
MASCOT	**Poor**	**Very Good**	**Average**	**Poor**
CODARTS	**Good**	**Average**	**Average**	**Average**
HOOD	**Good**	**Average**	**Average**	**Average**
ROOM	**Good**	**Good**	**Very Good**	**Average**
UML	**Excellent**	**Good**	**Average**	**Very Good**
UML-RT	**Average**	**Good**	**Good**	**Good**

score method (poor = 1, average = 2 , good = 3, very good =4, excellent = 5)

Table 3. Final Method Score

Method	Ranking Score
ROOM	24
UML	23
UML-RT	23
MASCOT	22
CODARTS	21
HOOD	19
CORE	18
YSM	17

Fig. 1 below depicts the final results for the methods/methodologies commonly used for real time software development. The results are obtained from the data in table 3.

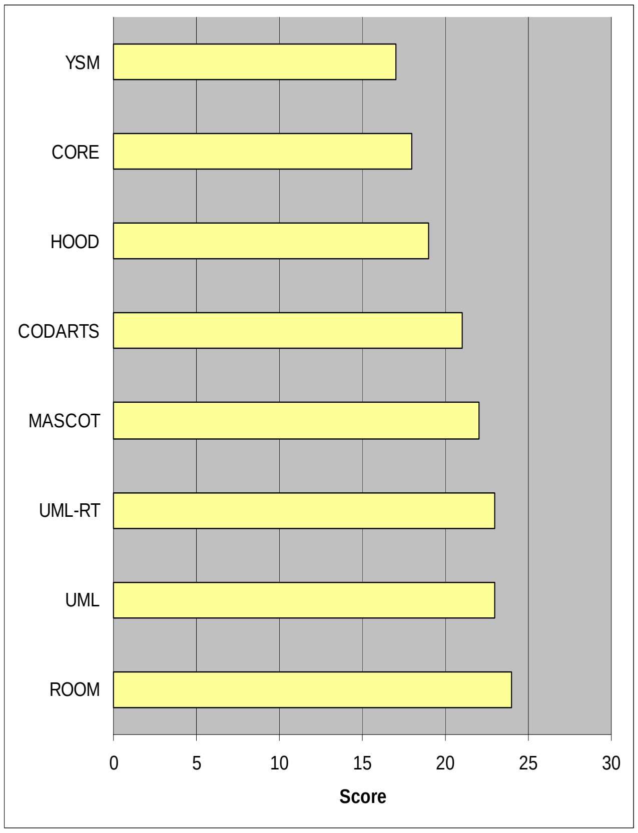

Figure 1. Real Time Method Ranking Bar Chart

Fig. 2 depicts how each method would actually fit in the SDLC (systems development lifecycle) process. The main steps included are requirements analysis, requirements specification, logical design, physical design, coding and testing. Obviously coding would imply integrating the components through interfaces etc. This is based on my own observations with reference to [3,4,7,8,9,12] and also practical use of some of these methods.

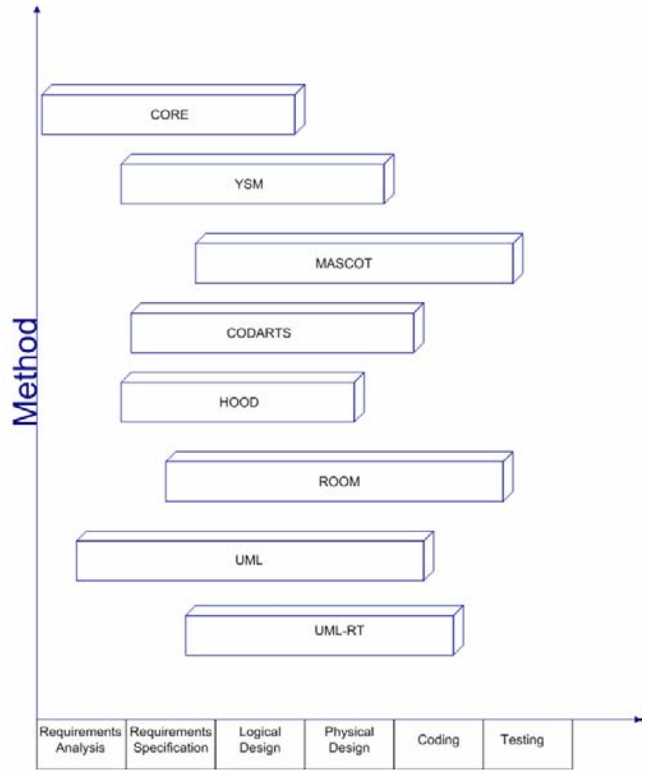

Figure 2. Methods vs SDLC phases

4. DISCUSSION

The results of this comparison in fig. 1 show that ROOM, UML and UML-RT rank as the three best methods for the development of RT systems. UML has the advantage of gaining widespread use and a lot of work is being done to improve UML continuously. 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. Another advantage of UML is that some UML diagrams are applied in a MDA approach and used to create PIM [6]. It is not justifiable that only one particular method is used. E.g. other methods like MASCOT embody principles that are still valid today and have been implemented in part in ROOM. UML does not have proper control flow diagrams similar to those found in YSM and CODARTS. 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. On the other hand CODARTS and YSM would be more suitable for things like avionics, a cruise control description etc. Another problem with the UML is that there are so many notations that it is often difficult to select what is really needed. E.g. Sequence diagrams and communication diagrams are semantically equivalent. When should one use one rather than the other? Also there are several ways in UML how to represent the same thing. Thus it is possible to have different diagrams of the same type representing the same scenario. In methods like the UML, ROOM, HOOD the messaging topology between objects is often 'loosely defined' with the possibility of having confusion.

It is obvious that what is lacking in one method might exist in another method. Object oriented methods are not a complete guarantee that there will be reusable components that will be available at a cheaper price especially if the interface needs to be rewritten. RT systems depend heavily on available hardware and might be operating system specific. This would imply that the design pattern is already biased from the onset of the project.

All the methods mentioned do not use proper formal verification techniques. Formal verification could be very important for checking that a design is free from deadlock. A lot of work is being done to try to formalize the UML like in [13]. There are also issues of performance analysis and task scheduling that need to be accounted for. CODARTS notations have already been used for performance analysis and task scheduling. The UML lacks performance analysis and does not take time into account. Actually the timing problem for many methods can be partially solved by translating dynamic UML diagrams into timed Petri Nets or using timed automata. The UML has been criticized by various authors. Note that even though ROOM is bound to a particular case tool its diagrams can easily be supported with other conventional case tools thus it has good case tool support.

The diagram in fig. 2 simply describes how each method fits in the systems development lifecycle process. These issues need to be considered when using these methods. If the focus is more on requirements engineering CORE could prove to be better than the others. CORE is the most adequate for requirements analysis and specification. HOOD, CODARTS and YSM cover part of the requirements analysis up to coding. MASCOT is more oriented towards the design, implementation phase and testing. The UML can be used for requirements analysis and can cover a wide aspect of the systems development lifecycle but it needs to be used in COMET or the USDP process. It could also be possible to combine CORE with UML. ROOM can cover up to testing depending on how it is used but it is not focused on the initial requirements analysis. What is evident is that no method covers all the required steps. This illustrates that for RT development one never be restricted to using a single method.

5. CONCLUSION

This paper has compared several methods for the analysis and design of real time systems. Although ROOM , UML, UML-RT stand out clearly as being the best on a number of attributes, in real life it is better not to be restricted to a single method. E.g. when students are using a method like the UML for their APT s it is always suggested that other notations from another method can be used. This is especially the case if there is a problem that requires some explanation. If using another notation or diagram would help then why not use it. A synonymous approach could be considered for industrial use. There are also other specific factors that need to be considered when selecting a method, like i) the type of industry involved, ii) user specialization, iii) if formal verification is required iv) reliability and safeness. It must be kept in mind that the results established in this paper are based on the set of attributes in table 1 & 2 might not be fully agreed upon by everybody.

6. REFERENCES

[1] Bennet, D. *Designing Hard Software*. Manning, 1996.

[2] Booch, G. *Object-Oriented Design with Applications*, 2d ed. Reading, Mass.: Addison-Wesley, 1991.

[3] Burns, A., Wellings, *A.. Real-Time Systems and Programming Languages*.Addison-Wesley, 2001.

[4] Cooling, J. *Software Design for Real-Time Systems*. Chapman & Hall, U.K.,1995.

[5] CORE, Controlled Requirements Expression (1986). *System Designers plc*, Fleet Hampshire, GU13 8 PD, UK document no.1986/0786/500/PR/0518.

[6] Coyle, F.P., Thornton, M.A. From UML to HDL: a Model Driven Architectural Approach to Hardware-Software Co-Design, *Information Systems: New Generations Conference (ISNG)*, Apr 2005, pp. 88-93.

[7] Gomaa, H. *Software Design Methods for Concurrent and Real-Time Systems*.Addison-Wesley, USA, 1996.

[8] Gomaa, H. *Designing Concurrent, Distributed and Real-Time Applications with UML*. Addison-Wesley, USA, 2004

[9] Graham, I. *Object Oriented Methods*. Addison-Wesley, USA 2000.

[10] Liu, J.W.S. *Real-Time Systems*, Pretence Hall, 2000.

[11] MASCOT, *The Official Handbook of MASCOT*. Joint IECCA and MUF Committee, 1987.

[12] Mullery, G.P., CORE - a method for controlled requirement specification. *,ACM International Conference on Software Engineering Proceedings of the 4th international conference on Software engineering*, Munich Germany 1979 , pp.126 – 135.

[13] Saldhana, J.A., Shatz, S.M., Hu, Z.Formalization of Object Behavior and Interaction From UML Models. *International Journal of Software & Knowledge Engineering*, 11(6), 2001, pp. 643-673.