World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:3, 2007

A Comparison of Software Analysis and Design Methods for Real Time Systems

Anthony Spiteri Staines

***Abstract*—**This paper examines and compares several of the most common real time methods. These methods are CORE, YSM, MASCOT, JSD, DARTS, RTSAD**,** ADARTS, CODARTS, HOOD, HRT-HOOD, ROOM, UML, UML-RT. The methods are compared using attributes like i) usability, ii) compositionality and iii) proper RT notations available. Finally some comparison results are given and discussed.

***Keywords*—**Software Engineering Methods, Method Comparison, Real Time Analysis and Design.

I. Introduction

REAL time analysis and development is an intricate process. This is because these systems exhibit special behavior. Real time timing, communication and reliability requirements are not easily explained and represented [6]. RT development possibly involves i) software engineering, ii) hardware engineering, iii) control engineering and iv) communication engineering. Minor changes to specifications could be very costly. Software programs embedded directly into hardware controller and devices might require entire rewriting. Hardware configuration problems exist when software engineers do not understand why specific processors, devices and operating systems have to be used.

Various methods and notations have been developed for the analysis and design of RT systems. These differ from normal methods because they focus on event driven behavior, communication and timing issues apart from static system properties. Methods are no guarantee that all software development problems will be solved. But they attempt to structure the analysis & development of RT systems applying design techniques and rules.

Methods and methodologies like CORE, YSM, MASCOT, JSD, DARTS, RTSAD are based on a data driven approach. Principles from traditional structured analysis and design are used [5], [8]. Methods like ADARTS, CODARTS, HOOD, HRT-HOOD, ROOM, UML, UML-RT use object oriented notations [1], [5], [9]–[10]. Initial object oriented methods lack the dynamic modeling principles found in the UML. The focus was more on static behavior and structural aspects.

Software design methods are different from software notations used for modeling or representing a system. A proper design method should focus on the software development lifecycle process. Software notations focus on very specific aspects of the design process. Some of these methods have been used for quite some time whilst others like the UML are quite modern.

A. Spiteri Staines is an assistant lecturer at the Department of Computer Information Systems, Faculty of Science, University of Malta, Msida, MSD 2080, Malta (phone: 00356-21373402; fax: 21312110; e-mail: toni_staines@yahoo.com, tony.spiteri-staines@um.edu.mt).

II. Explanation of Some of the Methods

A. Controlled Requirements Expression

The CORE method [5], [14]–[15] was designed in the UK for the requirements analysis phase and intended for the avionics industry. Core uses the following steps i) problem definition ii) define requirement viewpoints, iii) record viewpoints actions and data, iv) define the viewpoints iv) develop detailed models for each viewpoint and v) combine the single viewpoints into a composite model. CORE is suitable for the informal process of gathering the systems requirements expressed using informal notations like block diagrams and viewpoint diagrams. This approach could be used in conjunction with object oriented analysis. Control loops are available for use in the final diagram. 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 ii) it is unsuitable for architectural design iii) it is rigidly focused on several steps.

B. Jackson System Design

JSD [5] is different from YSM and other methods. This has been continuously improved upon from the original idea. The main steps are i) develop a model of the system, ii) produce a system specification, iii) implement as an executable design. The model tries to understand all possible events that will occur listing objects entities, actions, features, action order and updating mechanisms.

The model is used to build network diagram or specification. Using transformation rules the network diagram is systematically converted into a system implementation. This is known as transformation from specification to

implementation. The end result is having structured program charts or specifications that are easily implemented. It is possible to find CASE tools that produce source code from detailed program structure charts and program statements. Some possible problems with this approach is i) it has to be rigorously followed to the end to get results ii) it is difficult to combine with other approaches.

C. Real Time Structured Analysis and Design, Yourdon Structured Method and Design Approach for Real Time

RTSAD, DARTS and YSM (Hately-Pirbhai) explained in [3]–[5] include extensions to DFDs [7]–[8]. These add details for event flows and control transformations like discrete, continuous, triggered, enable/disable etc. These methods also use state transition diagrams.

In a RTSAD approach [3]–[4] the following steps are normally carried out. i) The system context diagram is developed, ii) data flow/control flow decomposition is performed, iii) control transformations or control specifications are built, iv) process specifications are defined and v) the data dictionary is developed. This approach decomposes a system into many sub components. Some resulting diagrams are data flow/control diagrams and state transition diagrams supported by mini-specifications.

The YSM described in [5] and [8] 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.

The DARTS method [4] has the following steps i) develop system specification using RTSAD notations, ii) structure the system into concurrent tasks, iii) define task interfaces, iv) design each task. For steps ii-iv task architecture diagrams and structure charts are obtained from the control flow diagrams in step i. This is similar to some of the steps in the YSM.

The main idea is to model the system control flows correctly and develop it further. It is possible to derive program code from the decomposed system models and task architecture diagrams.

Some advantages of these methods are i) highly structured data analysis is used. Limitations are i) they are unsuitable for prototyping. ii) steps must be followed sequentially for successful implementation iii) It is possible to take a long time to implement the complete system.

D. Modular Approach to Software Construction, Operation and Test

MASCOT [5], [13] was first issued in 1970s by the Royal Signals and Radar Establishment UK and successive versions MASCOT 3 exist. 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 real time 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) the structure of single thread processes is transformed. iv) components are coded 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 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 rapid application development, iv) it is expensive to apply.

E. Ada based Design Approach for Real Time

ADARTS [4] is a modified version of DARTS mainly intended for use with the ADA language. The structured design step from DARTS is replaced with information hiding module structuring step. It can be considered to be similar to DARTS but with some improvements as regards information hiding, abstraction and decomposition.

F. Concurrent Design Approach for Real Time

CODARTS includes many improvements from DARTS [4]. CODARTS classifies message passing into several types not normally found in other methods. The diagrams used are

similar to control flow diagrams. Special symbols are included for different types of message communication e.g. loosely-coupled message communication, tightly-coupled message. Available diagrams are task architecture diagrams, software architecture diagrams and STDs. These are easily implemented in ADA. Some limitations of CODARTS are i) Designed mainly for the ADA language. ii) Notations used are not well understood iii) complex to use iv) uses a limited number of views.

G. Real Time Object Oriented Modeling

ROOM is an object oriented method that uses ROOMchart diagrams based on an 'actor' concept. ROOMcharts are similar to the UML statechart diagrams. The ROOM method is more oriented towards the actual implementation and physical design. It has a limited number of diagrams for the initial requirements engineering. A limitation of ROOM is that it requires a particular CASE tool called 'Objectime'. It is possible to generate C++ code from diagrams. The actor initiates a sequence of events. Ports are used for communication, threads control behavior. Limitations of ROOM are: i) tied with one particular CASE tool called 'ObjecTime' ii) a limited number of diagrams showing only certain views are available.

H. Hierarchical Object Oriented Design

HOOD is one of the first object oriented design methods. It is mainly aimed at using ADA. It can be useful for prototyping. The idea is to identify objects in a parent to child relationship and their operations. A graphical system description is produced in a control/ dataflow diagram. This explains the flow of information between a set of objects. Diagrams can be decomposed to the required levels. The Top-Level Object is normally an active object using the lower-level. Rules for passive objects and active objects exist. Limitations of HOOD are: i) does not distinguish Data Flows between Objects from Event Signals ii) is not so simple and straightforward to use iii) only one main diagrammatic type is used. HRT-HOOD implements some improvements over HOOD.

I. Unified Modeling Language

The UML [1]–[2], [9]–[11] 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 as outlined in [1] 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.

In UML the focus is on modeling a system rather than on managing the software development process. This implies that the UML should be used in a framework like the USDP (Unified Software Development Process) created by the OMG or COMET (Concurrent Object Modeling architectural design method) [11].

J. Unified Modeling Language – Real Time

UML-RT [17] 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.

III. Method Comparison

The methods have been compared using three fundamental issues and data from [1]–[17]. These are i) usability, ii) compositionality and iii) Proper RT notations available. Some other issues are presented in [7], [10].

A. 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.

B. Compositionality

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.

C. Proper Real Time Notations Available

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. Special notations have to be used distinguish real time event types. Resource management support refers to processing loops, scheduling, activity management, control management and performance management for the composite system.

D. 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.

TABLE I
USABILITY SCORE

Method	Ease of use	Clarity of Diagrams/ notations	CASE tool support
CORE	3	4	4
JSD	3	4	4
YSM	3	4	4
RTSAD	2	3	3
DARTS	2	3	3
ADARTS	2	3	3
CoDarts	2	3	3
MASCOT	1	2	1
HOOD	3	3	3
ROOM	2	3	3
UML	5	4	5
UML-RT	3	3	2

The usability explains how easy the method is.

TABLE II
COMPOSITIONALITY SCORE

METHOD	Diagram Consistency	Recursion	Support for Graphical to Textual Conversion	Cross-references
CORE	3	3	2	4
JSD	4	4	4	4
YSM	3	3	3	4
RTSAD	4	3	3	4
DARTS	4	3	4	4
ADARTS	4	4	4	4
CoDarts	4	4	4	4
MASCOT	4	4	4	4
HOOD	3	2	2	2
ROOM	4	3	4	3
UML	1	1	2	1
UML-RT	3	3	3	2

Compositionality is based on identifying how the notations, diagrams, references and textual descriptions properly fit together.

TABLE III
PROPER REAL-TIME NOTATIONS SCORE

METHOD	Message Comm.	Support for RT Resource Management	Support for Timing Requirements	Support for Special Event Types
CORE	1	1	1	4
JSD	3	3	2	4
YSM	1	2	1	4
RTSAD	3	2	2	4
DARTS	3	3	2	4
ADARTS	3	4	2	4
CoDarts	4	4	2	4
MASCOT	4	4	2	4
HOOD	2	3	2	2
ROOM	3	4	4	3
UML	2	1	1	1
UML-RT	3	2	2	2

Proper real time notations refers to how well does the method support constructs that identify and classify real time behavior.

IV. RESULTS

The usability results indicate that UML is the best usable method followed by CORE, JSD and YSM.

The compositionality results indicate that methods like MASCOT, CODARTS, RTSAD have better compositionality. This is because of good cross-referencing between notations. There is also the fact that these methods 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.

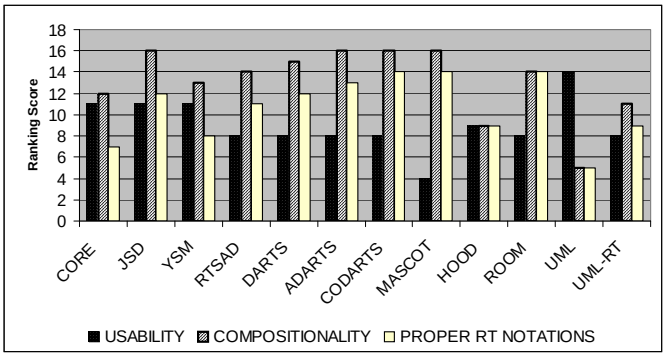

Fig. 1 Usability, Compositionality and Proper RT Notations Score

V. 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. 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. Another advantage of UML is that some UML diagrams are applied in a MDA approach and used to create PIM [6]. 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 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. The UML has given the initiative to create other modeling concepts and methods like AGILE and FMCs(Fundamental modeling concepts).

Methods like JSD, YSM, DARTS, ADARTS, MASCOT CODARTS, HOOD 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.

References

[1] S. Bennett, J. Skelton, K. Lunn, *UML*. Schaums Outline 2nd ed., New York: McGraw-Hill, 2005, pp. 5–18.
[2] P. Roques, *UML in Practice*. UK: Wiley, 2005, ch. 1. & ch. 2.
[3] R. Williams, *Real-Time Systems Development*. UK: ELSEVIER, 2006, ch. 11.
[4] H. Gomaa, *Software Design Methods for Concurrent and Real-Time Systems*. Addison-Wesley, 1996, pp.137-294.
[5] J.E. Cooling, Software Design for Real-Time Systems. London: Chapman & Hall, 1995, ch. 10.
[6] J.W.S. Liu, *Real-Time Systems*. NJ: Pretence Hall, 2000, ch. 1.
[7] D.C. McDermid, *Software Engineering for Information Systems.* GB: McGraw-Hill, 1990, ch.1. ,ch. 2.,ch.6. & ch.10.
[8] S.Goldsmith, *A practical guide to Real-Time Systems Development*. Hetrfordshire: Pretence Hall, 1993, ch. 1.
[9] S.R. Schach, *Introduction to Object-Oriented Analysis and Design with UML and the Unified Process*. NY: McGraw-Hill, 2004, ch.3. & ch. 11.
[10] I. Graham, *Object-Oriented Methods Principles & Practice*. ED: Pearson Education, 2001, ch.6.
[11] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML. IN:Addison-Wesley, 2001, ch. 2.
[12] D. Bennett, *Designing Hard Software The Essential Tasks*. Greenwich: Manning, 1997, pp. 25-100.
[13] *The Official Handbook of MASCOT*, Joint IECCA and MUF Committee, 1987.
[14] G.P. Mullery, "CORE - A Method for Controlled Requirement Specification", *Proceedings of the 4th international conference on Software engineering*, Munich Germany 1979 , pp.126 – 135.
[15] *CORE Controlled Requirements Expression*, System Designers plc, Fleet Hampshire, GU13 8 PD, UK document no.1986/0786/500/PR/0518, 1986.
[16] B. Selic, G. Gullekson, P.T. Ward, *Real-Time Object-Oriented Modeling*. Wiley, 2005, ch.1. & ch. 2.
[17] M. Antonsson, P. Hansson, "Modeling of Real-Time Systems in UML with Rational Rose and Rose Real-Time based on RUP", Ericsson Mobile Data Design.., AB (ERV) Gothenburg, Sweden, M.S thesis open rep. ERV/G-01:071 Uen,2001.