A Metrics Driven Design Approach for Real Time Environment Application

Mahmood Ahmed and Muhammad Shoaib
Department of Computer Science and Engineering, University of Engineering and Technology Lahore, Pakistan

Abstract: *Design of real time environment application is the most exigent task for the designers comparing to non Real Time Application (RTA) design. The stringent timing requirement for task completion is the problem to handle at design time. The design complexity is increased manifolds when object oriented design methods are used and task deadlines are introduced at design stage. There are many design methodologies available for the real time systems but as far as the researcher is concerned none addresses all the problems of real time system design specially the issues of deadline inheritance and dynamic behavior of system if deadlines are introduced at early stages of the design. Most of the methodologies leave the task of handling the timing constraints for the implementation phase at the programming language level. In this paper we have proposed a design approach incorporated with our novel design metrics verification for measuring the design of real time environment applications. The metrics are measured for design of a real time weapon delivery system and it is illustrated that how design quality can be assessed before implementation.*

Keywords: *Deadlines, timed state statecharts, design metrics, real time systems.*

Received November 26, 2011; accepted June 11, 2012

1. Introduction

A Real Time Application (RTA) is one that takes into consideration the constraints like: strict timing limit on response of the system, normally it has event driven scheduling, low-level programming, software highly coupled to particular hardware, committed dedicated function, the computing system might be within a control loop, variables are normally volatile, multi-tasking is often implemented, scheduling demand is run-time, environment is also unpredictable, continuously running system is requirement, and is used as life-critical applications [18, 34]. RTA design is challenge due to the difficulty in incorporating timing information of various tasks in the design architecture [3, 7, 13, 14]. Most designers left this deadline management as an extra task for the developer to handle, in the implementation phase [10]. There are problem not addressed in these methodologies e.g., adding a deadlines to even simple automata makes it highly complex [21]. Properties of simple automata become hard to prove when time constraints are introduced. Most of the methodologies do not support inheritance of deadlines [10, 13, 23].

2. Related Work

There are many design methodologies proposed for real time system design. Some of them are briefly described. Jackson [13] developed Jackson System Development (JSD) which is a linear software development methodology. Main goal was to map progress of the system to be modeled with the progress in the real world. Timing is considered only at the 5th step is JSD method. A Real-Time operating System (ARTS) developed in the ART project at carnegie Mellon University targets the real time systems and it is an object oriented [20]. This methodology is based on the RTC++ which is real time extension of the C++ [12]. Behavior of the objects in ARTS has no clear understanding that how it is modeled.

Concurrent Object-Based Real-time Analysis (COBRA) is a mix of concepts of Object-Oriented Analysis (OOA), JSD [13, 20, 24] and real time structured analysis [10]. It uses the notation of state diagram and real time structured analysis. For distributed environments COBRA has an advantage due to its support for decomposition approach. One drawback of COBRA is that it does not consider deadlines.

HOOD/PNO is another methodology introduced by [23]. Hierarchical Object-Oriented Design (HOOD) is defined by European Space Agency [8]. Petri Net Objects (PNO) is a way to illustrate the behavior and control structure of objects using Petri nets. The entire life cycle including design, analysis and implementation is dealt in this methodology. It covers the life cycle from requirements to code. The limitations of this methodology are that it does not tackle concurrency directly. Deadlines of objects are not dealt in design but are left as implementation challenge for the language. Hard Real Time Hierarchical Object Oriented Design (HRT-HOOD) [4] is adapted from HOOD for real time environments [20]. Abstraction is the main focus of this methodology. Deadlines are better conceptualized due to abstraction. HRT-HOOD separates the high level design activity into sections. HRT-HOOD supports

five kinds of objects including passive, active, protected, cyclic and sporadic. This methodology does not clarify if concurrency is processors and if threads are inside an object. supported when objects are assigned to physical processors and if threads are inside an object.

OCTOPUS methodology is for handling the embedded real time systems [33]. OCTOPUS extended the Object Modeling Technique (OMT) [25] for catering synchronization, concurrency, interrupts, end-to-end response time, hardware interfaces and communication. This methodology uses state charts for behavior modeling. Concurrency and deadline management are handled. Just like OMT inheritance is also supported by OCTOPUS [20]. The drawback is that it does not support the full life cycle but only design and implementation phases. OMT is a general methodology and not a real time methodology. Real time Object Oriented Modeling (ROOM) methodology uses two concepts i-e abstraction level and dimension [3]. Based on the nature of the problem the dimension model partitions the system. The system is then partitioned into three levels of abstractions i-e system level, concurrency level and detail level. Daponte *et al*. [6] author claims that Real Time Objects (RTO) is suited for hard real time programming. This methodology does not allow concurrency between the objects. Transnet [26] is another proposed extension to design methodologies for real time systems. To model the behavior and for verification this methodology uses Petri nets as in HOOD/PNO that as uses Petri nets. This methodology focuses not only on functionality of design but also concerns about deadlines, message passing and concurrency of the objects. Measurement is becoming the most important factor in software engineering because if you cannot measure you cannot control the progress of development [1, 9].

3. Proposed Design Method

Most of the methodologies developed are centered around the philosophy of using formal technique that are in practice for many years. But as we introduce the concept of time the simple automata becomes so complex even for medium size problems [3]. To measure the design problems associated with timing constraints or deadlines in real time systems we have proposed the following design technique shown in Figure 1. This research paper is extracted from [1].

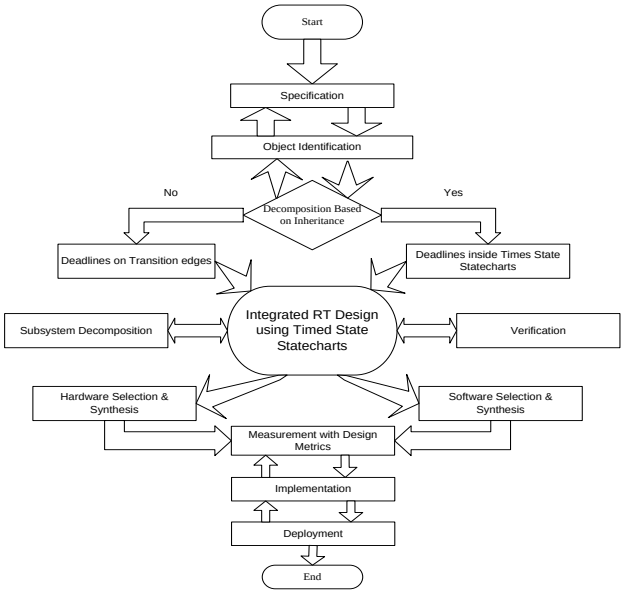

Figure 1. Proposed design method flow for real time system steps of the design approach.

3.1. Steps of the Design Approach

System Requirements Specifications (SRS) is the "what" part (mean what is the problem) which is a logical document. It specifies the system requirements without dictating how those requirements must be implemented. Design is the "how" part (How the problem should be addressed and how it should be solved) is the first phase in which we make a transition towards the solution. The goal during the design phase is to produce correct designs [15].

3.2. Specification

The specification step/phase shown in Figure 2 is accomplished two stages i-e problem conception stage and the decomposition stage [17].

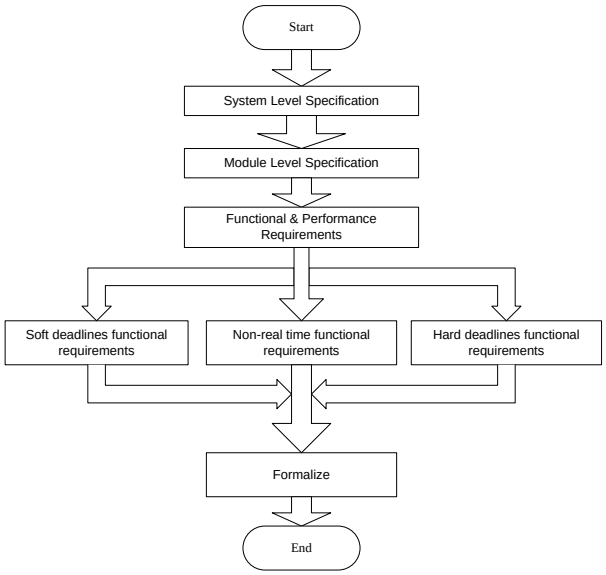

Figure 2. Specification phase.

The real time application to be developed is described. The operational environments of application

and its functionality, constraints and tolerances are given a formal shape. The functional and performance requirements for mechanical and electrical hardware, sensors and actuators and components related to control and operation of the real time environment application. Identify all the timing constraints and separates them into hard, soft and firm timing constraints. Hard deadline mean that if the constraints are not met the result is a catastrophic failure or an accident. Soft deadline mean that the results may be invalid and the repetition of a task must be done before the next task can take place. In case of firm deadlines failure there is no catastrophic failure but it might be serious.

Identify all of the possible tasks the real time system needs to perform and categorize them into PERIODIC, synchronous, asynchronous and sporadic tasks.

3.3. Object Identification

System requirements are broken-down into a group of appropriate objects and Real-Time Objects (RTOs) and are distinguished through a decomposition scheme such as Multi Dimensional Decomposition (MDD) [17]. From the specifications the objects relating to sensors, actuators and control are identified shown in Figure 3. The control objects are placed in-between the sensor objects and the actuator objects.

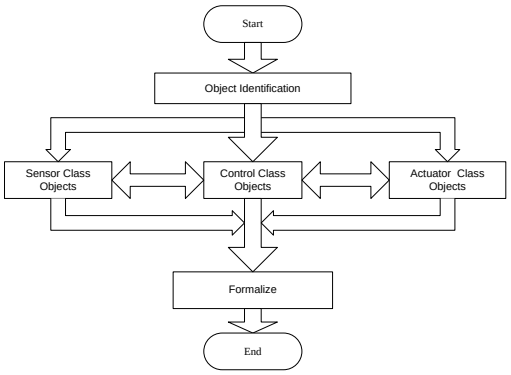

Figure 3. Object identification phase.

3.4. Decomposition

Decomposition is very crucial in the design because the decision taken in this step will be the deciding factor on which technique is based. Identify the objects that are to be inherited and the objects that are not to be inherited. If an object is to be inherited, the deadline must be included inside the state chart. This is to ensure that there should be no problems when deadlines are also inherited along with the other features. If an object is not to be inherited than its deadline may be indicated in the transition as in regular formalism methodologies. This will impart the characteristic of formal design approach. Identify superstates and substates based on selective inheritance. To model concurrency inherit all the concurrent tasks as substates and must be contained in superstate. Most of the methodologies use the formal techniques to represent timing conditions on the transition arcs or edges. The drawback of this technique is that it is difficult to describe the behavior of the deadlines when they are inherited. The objects that are to be inherited will be designed using the timed state statecharts which is an informal representation of the dynamic behavior. Identify superstates and substates based on selective inheritance and design the hierarchy. To model concurrency inherit all the concurrent tasks as substates and must be contained in superstate. The selective inheritance is used to separate objects so as to use design in two different ways. The purpose is to take maximum advantage of both the formal design methodologies and informal methodologies.

3.5. Integrated Real Time System Design

3.5.1. Architectural Design

The objective of software architecture is to provide the mainly elementary foundation for convince about design decisions and set up important work breakdown structures [22]. Architectural document makes it possible to understand a complex system like real time system. Architecture is an overall perspective of the system design. Before you go on to detailing the subsystems you want to grasp or understand the overall design of the software. The architectural design is necessary because to grasp the system from bottom up approach is very difficult with thousands of classes and components. Architecture also facilitates reuse. Design the software architecture's module view, component and connector view, and allocation view [15].

3.5.2. Detail Design

For detail design of real time system the timing constraints must be considered at the class and component level, only then the methodology is considered as true real time system methodology.

Throughout the design phase we recommend to use the Harel's Diagrams (also called the statecharts) [11] to describe the dynamic behavior of the system modules. To model the deadlines the theory of timed automata [3] and the concept of timed state statecharts is used. Then the design must be integrated showing links to various modules. If the modules are not related they are just a collection of library and not a collective system as a whole.

Consider the design example of the Coolant spill of nuclear reactor shown in Figure 4. This simple design is drawn using the Statechart+ solution of the Wang and Chen [32] along with the Timedstate statechart solution shown in Figure 5.

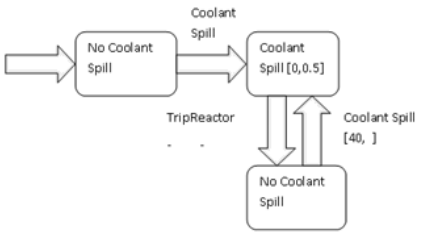

Figure 4. State chart + solution.

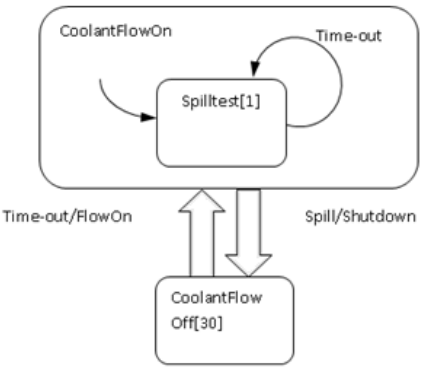

Figure 5. Timedstate state charts solution.

The Wang and Chen [32] solution uses both timed transitions and timed states that leads to upper time bound ambiguities and the lower bound time ambiguities. To make it simple just use the timed states and timer the solution becomes simple and upper and lower bound time ambiguities are resolved. The same solution is presented as follows. This simple modification can resolve the ambiguities that arise due to inheritance of state chart objects having timing information on transition edges.

3.6. Hardware Software Selection

Normally in real time system the hardware is tightly coupled with the software. The single or multiprocessor architecture determines the true degree of parallelism [16]. The hardware selection depends on the requirements specification, e.g., in case of concurrent processing requirement a single processor will only give pseudo parallel execution not the true parallel execution as in case of multiprocessor system [16]. This phase is also very important because numerous hardware and software are available in the markets that claim real time capabilities. It is not feasible to design the complex real time applications from scratch. There are dedicated hardware's for data acquisition from the real time environment and control hardware. There are also real time operating systems available. The very first examples of real-time operating systems for a large-scale projects were, the IBM and american airlines's Transaction Processing Facility (TPF) which was built for the Sabre. At present the top most known, real-time operating systems are Windows CE, OSE, RTLinux, LynxOS, QNX, VxWorks [33].

There is misconception that high level languages are not recommended for stringent timing requirements and low level languages like assembly is recommended. But it is one of the misconceptions pointed out in [29, 31]. According to [6] the actual point of concern should be that if the language you chose allows access to low level hardware interface without the extra run time support penalty.

3.7. Verification using Proposed Design Metrics

We have defined the following eight new metrics for measuring a real time system design [1]. Purpose is to measure the design before implementation. Object oriented design metrics [5] but for measuring real time application design, no metrics are available.

3.7.1. Soft Deadline Cohesion Factor

It is defined as the ratio of the classes having soft deadlines to the total no. of classes having soft, hard, overridden deadlines.

$$SDCF = \sum_{n}^{n}\sum_{j}^{m}\frac{C_{S_{ij}}}{C_{s_{ij}} + C_{h_{ij}} + C_{o_{ij}}} \quad (1)$$

Where
n=Total number modules constraint by timing restriction.
m=Total number classes per module constraint by timing restriction.
$C_{sij}=i^{th}$ class in the j^{th} module with soft deadlines.
$C_{hij}=i^{th}$ class in the j^{th} module with hard deadlines.
$C_{oij}=i^{th}$ class in the j^{th} module with overridden deadlines.

This factor tells about how cohesive the modules are in relation to soft deadlines. Higher the factor mean the system modules may be given less concentration because of error tolerance level is more in that module.

3.7.2. Hard Deadline Cohesion Factor

It is defined as the ratio of the classes having hard deadlines to the total number of classes having soft, hard, overridden deadlines.

$$HDFC = \sum_{n}^{n}\sum_{j}^{m}\frac{C_{h_{ij}}}{C_{s_{ij}} + C_{h_{ij}} + C_{o_{ij}}} \quad (2)$$

Where
n=Total number modules constraint by timing restriction
m=Total number classes per module constraint by timing restriction
$C_{sij}=i^{th}$ class in the j^{th} module with soft deadlines.
$C_{hij}=i^{th}$ class in the j^{th} module with hard deadlines.
$C_{oij}=i^{th}$ class in the j^{th} module with overridden deadlines.

This factor tells about how cohesive the modules are in relation to soft deadlines. Higher the factor means

the system modules may be given more concentration because of error tolerance level for these modules are very low.

3.7.3. Overridden Deadline Class Factor

It is defined as the ratio of the classes having overridden deadlines to the total no. of classes having soft, hard, overridden deadlines.

$$HDFC = \sum_{n}^{n}\sum_{j}^{m}\frac{C_{O_{ij}}}{C_{s_{ij}} + C_{h_{ij}} + C_{o_{ij}}} \quad (3)$$

Where
n=Total number modules constraint by timing restriction.
m=Total number classes per module constraint by timing restriction.
$C_{sij}=i^{th}$ class in the j^{th} module with soft deadlines.
$C_{hij}=i^{th}$ class in the j^{th} module with hard deadlines.
$C_{oij}=i^{th}$ class in the j^{th} module with overridden deadlines.

This factor is the most important because it tells about deadline related ambiguities that lies in those modules having high Overridden Deadline Class Factor (ODCF) value. These ambiguities are due to the inheritance of deadlines. Most of the concentration must be given to those modules having high ODCF.

3.7.4. Soft Overriding Factor

The overriding factor is defined as the ratio of overridden classes to the total no. of classes having hard deadlines.

$$SOF = \frac{\sum_{o=1}^{n_o} C_o}{\sum_{b=1}^{n_h} C_h} = \frac{ODFC}{HDFC} \quad (4)$$

This factor tells the overall trend the module towards soft or the hard real time approach. A value less than 1 means timing constraints have to be met all cost.

3.7.5. Message Exchange Factor

Number of exchanged messages considered per second between project partitions.

$$MEF = \frac{\sum_{i=1}^{n_e} m_i}{T} \quad (5)$$

Higher the MEF more critically that module must be analyzed.

3.7.6. Early Decomposition Factor

The Early Decomposition Factor is defined as the ratio of no. of project partitions to the project stage no. times the message exchange factor.

$$EDF = \frac{(Number\ Of\ Partition\ Of\ Project)}{(Project\ Stage\ Number)} \times (Message\ Exchange\ Factor)$$

Mathematically it is represented as:

$$EDF = \frac{N_p}{S_n} \times \frac{\sum_{i=1}^{n_e} m_i}{T} \quad (6)$$

If early partitioning into sections for a large system is done then it could lead to a poor design if at a later stage it is found out that message traffic between different sections of the system will consume enormous amount of resources. It is also kept in mind that this metric has not as much of importance when only object oriented systems are under consideration.

3.7.7. Deadline based Predictability Factor

The Deadline based Predictability Factor is defined as ratio of the no of classes with soft deadline to the total this factor is defined as number of sub classes plus total no. of multithreaded objects.
Mathematically it is represented as:

$$DBPF = \frac{\sum_{o=1}^{n_o} C_o}{\sum_{b=1}^{n_b} C_b} + \sum_{1}^{n} Obj_{mt} \quad (7)$$

Ideally the first factor should be than 1 and practically be close to zero. The second factor should be more than 1, because multithreading increases the predictability [20].

3.7.8. Life Cycle Support Factor

Life Cycle Support Factor (LCSF) is the ratio of number of phases having support for deadlines to the total number of phases on the life cycle plus one.

$$EDF = \frac{(No.\ Of\ Phases\ having\ deadline\ support)}{(Total\ No.\ of\ phases\ in\ the\ life\ cycle) + 1} \quad (8)$$

Every methodology has support for Software life cycle in some phases. If this factor is equal to 1 this means that the methodology supports the entire life cycle beyond the code release and into the code maintenance.

3.8. Implementation

In this phase we take the deliverables/documents produced during the requirements phase and design phase and implements them using suitable tools and technologies. Test cases are accomplished and prepared/automated in case of validation testing. In general, an extensive amount of testing is also performed on the early system versions during this phase, not only to validate the system, but to validate that there are no anomalies in the test cases themselves. It is also necessary to take into account the implementation languages like RTC++ or Ada 9X may do the job. Along with the general purpose languages like, Ada, Modula, and Java there are also special purpose languages for real time systems like Esterel, Lustre, Signal and Statecharts [29]. The implementation may be done in any of the modeling language. The Unified Modeling Language (UML) is a popular choice for modeling of real time system [19].

It will help you measure metrics automatically through available tools [27].

3.9. Deployment

Deployment starts with understanding the client personals including system users, system operators, support staff and system owner as shown in Figure 6. Discuss deployment plan with key personals and modify if needed. After your plan is vetted start installation and onsite testing.

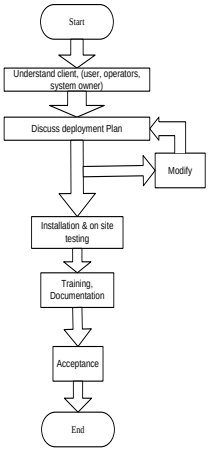

Figure 6. Deployment phase.

4. Case Study

We have considered the design of weapon firing system example [7] as our case study to experimentally evaluate the defined metrics. For calculation of other design metrics consider the behavior model of classes in concurrent operation as shown in Figure 7.

$$SDFC = \frac{1}{1+1+1+1+1+1+1+1} = \frac{1}{8} = 0.125 \quad (9)$$

The value for this factor is towards a lower side meaning that the this module has 12.5% soft deadline class objects and the module design can be considered as tilting towards a soft real time class of applications. This indicates that the predictability of the module is towards higher side. Only once class objects has 1000 millisecond deadline which is achievable by most of the currently available hardware software systems in the market.

$$HDFC = \frac{1+1+1}{1+1+1+1+1+1+1+1} = \frac{3}{8} = 0.375 \quad (10)$$

The value for this factor is also towards a lower side meaning that the this module has 37.5% hard deadline class objects and the module design can be considered as tilting towards a soft real time class of applications. Higher the Hard Deadline Cohesion Factor (HDFC) factor means that the predictability is hard to guarantee.

The ODFC factor for this module is zero. This indicates that there are no class objects having overridden soft deadlines with the hard deadlines. This module's complexity is zero.

$$ODFC = \frac{0}{1+1+1+1+1+1+1+1} = \frac{0}{8} = 0 \quad (11)$$

SOF is measured as: $SOF = 0/0.375=0$.

The MEF factor is calculated as:

$$MEF = \frac{23}{1000} = 0.023\frac{msg}{ms} \quad (12)$$

The number of messages exchanged per millisecond are not on the higher side for this module. The system resources are sufficient to ensure the predictability of all the tasks to be completed in specified time limit.To calculate the EDF factor we use the Figures 7 and 8. There are 11 partitions in this weapon delivery system and at this stage the project stage number is 2 so the metric is calculated as

$$EDF = \frac{11}{2} \times \frac{13}{1000} = 0.0715 \quad (13)$$

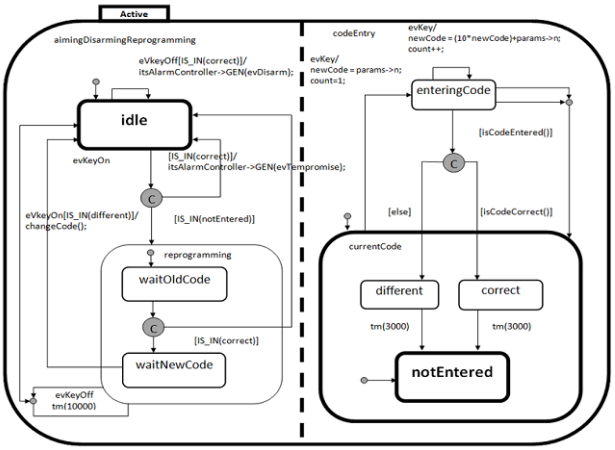

Figure 7. Concurrent State threads with timing info [7].

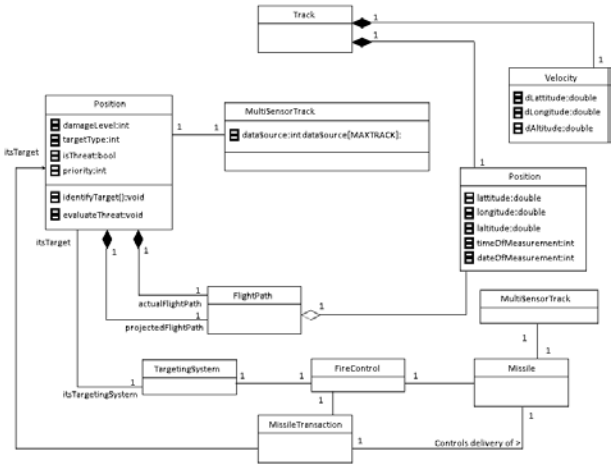

Figure 8. Weapon delivery system Modules [7].

The value of this metric is normal and it is not too early to partition the project. It will not consume too much system resources and the decision to partition at this stage is normal.

$$DBPF = \frac{1}{8} + 2 = 2.125 \quad (14)$$

For module of this size the number of multithreaded objects is normal but there should be more multithreaded objects for the predictability to increase.

Now for Life Cycle Support Factor (LCSF) metric verification we consider the different methodologies, as shown in the Table 1, and calculate the LCSF metric for them.

We find out that which phases have support for the deadlines in each of

the methodologies and calculate the LCSF factor for each of the methodology. To simplify the case we have considered the following precise number of phases for deadline support consideration.

Table 1. Life cycle support factor for different methodologies [8, 14, 20, 24, 25].

Methodology	Specification	Design	Implementation	Testing and verification	Deployment	Maintenance	Phases having Support	LCSF
JSD	N	Y	N	N	N	N	1	0.1429
ATRS	Y	Y	Y	N	N	N	3	0.4286
COBRA	N	N	Y	N	N	N	1	0.1429
HOOD/PNO	N	N	Y	N	N	N	1	0.1429
HRT-HOOD	Y	Y	Y	Y	N	N	4	0.5714
OCTOPUS	N	Y	Y	N	N	N	2	0.2857
OMTs	N	N	Y	N	N	N	1	0.1429
ROOM	Y	Y	Y	N	N	N	3	0.4286
RTO	N	N	Y	N	N	N	1	0.1429
MDTRA	Y	Y	Y	Y	Y	N	6	0.714286
Transnet	N	N	Y	N	N	N	1	0.1429

After thoroughly studying these methodologies we were able to come to the fact that not a single methodology has full life cycle support for the deadlines. Most of the methodologies support deadlines only at the implementation phase or the programming language. A 3-D plot of the above table is shown Figure 9 to visualize the above mentioned fact. From the plot it is clear that arts, hrt-hood, room and octopus have better support for the deadlines in various phases of the software development life cycle.

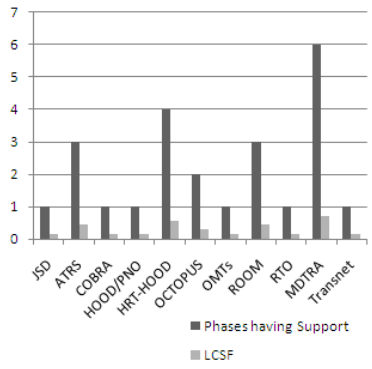

Figure 9. 3-D Plot of LCSF factor.

$$LCSF_{JSD} = \frac{N+Y+N+N+N+N}{6+1} = \frac{0+1+0+0+0+0}{7} = 0.142857 \quad (15)$$

$$LCSF_{ARTS} = \frac{Y+Y+Y+N+N+N}{6+1} = \frac{1+1+1+0+0+0}{7} = 0.428571 \quad (16)$$

$$LCSF_{COBRA} = \frac{N+N+Y+N+N+N}{6+1} = \frac{0+0+1+0+0+0}{7} = 0.142857 \quad (17)$$

$$LCSF_{HOOD/PNO} = \frac{N+N+Y+N+N+N}{6+1} = \frac{0+0+1+0+0+0}{7} = 0.142857 \quad (18)$$

$$LCSF_{HRT-HOOD} = \frac{Y+Y+Y+Y+N+N}{6+1} = \frac{1+1+1+1+0+0}{7} = 0.571429 \quad (19)$$

$$LCSF_{OCTOPUS} = \frac{N+Y+Y+N+N+N}{6+1} = \frac{0+1+1+0+0+0}{7} = 0.285714 \quad (20)$$

$$LCSF_{OMT} = \frac{N+N+Y+N+N+N}{6+1} = \frac{0+0+1+0+0+0}{7} = 0.142857 \quad (21)$$

$$LCSF_{ROOM} = \frac{Y+Y+Y+N+N+N}{6+1} = \frac{1+1+1+0+0+0}{7} = 0.428571 \quad (22)$$

$$LCSF_{RTO} = \frac{N+N+Y+N+N+N}{6+1} = \frac{0+0+1+0+0+0}{7} = 0.142857 \quad (23)$$

$$LCSF_{MDTRA} = \frac{Y+Y+Y+Y+Y+N}{6+1} = \frac{1+1+1+1+1+0}{7} = 0.714286 \quad (24)$$

$$LCSF_{TRANSNET} = \frac{N+N+Y+N+N+N}{6+1} = \frac{0+0+1+0+0+0}{7} = 0.142857 \quad (25)$$

In this calculation each ‘N’ is given 0 (zero) value. It means that the methodology has no support in that phase. Each ‘Y’ is given value 1 (one) which indicates that the methodology has support for deadlines in that phase.

Most of the methodologies leave this task of handling the issues of deadlines to be handled by the developers at the implementation phase. Unfortunately we are not able to find real time system design examples that have considered the timing constraints in the entire life cycle. So we are unable to measure the values for most of the metrics to evaluate the design for quality.

For calculation of design metrics through tool SDMetrics [27] is a good choice. It is a software design metrics tool for the UML diagrams. UML is becoming the favorite software design tool for most of the designers.

5. Results of the Study

The results of our study reveal that no real time system’s design methodology have support deadlines for the entire life cycle. For methodology to be considered as true real time system design methodology, it must have to address the issues relating to the deadlines as early as possible in the software development life cycle and ideally in the entire life cycle. The actual practice is that deadlines are left as task for the developers to handle at the

programming language level during the implementation phase. We calculated the LCSF metric for the various design methodologies manually and results are plotted. We found that high value of LCSF is required for a methodology to be closer to a true real time methodology. To follow our approach the design must be in object oriented approach and the timing constraints must be introduced at the class and object level. Then that design should be measured using our defined metrics.

6. Conclusions

In this paper we have proposed a metrics driven design approach for real time environment applications and also suggested a slight modification in the design technique when using state charts to resolve the ambiguities that arise due to inheritance of state chart objects having timing information on transition edges. We have also incorporated the methodology with the additional phase which we named as verification by design metrics. Eight design metrics have been defined to test the design for identifying the areas where a more thorough concentration is required. Those areas or the modules are studied again and the ambiguities are resolved.

References

[1] Ahmed M., "Design Quality Metrics for Real Time Environment Application," Ph.D. Dissertation, Department of Computer Science and Engineering, University of Engineering and Technology, Lahore-Pakistan, 2012.

[2] Ahmed M. and Shoaib M., "Novel Design Metrics to Measure Real Time Environment Application Design," *Journal of American Science*, vol. 7, no. 7, pp. 222-226, 2011.

[3] Alur R. and Dill L., "The Theory of Timed Automata," *Theoretical Computer Science Jornal*, vol. 126, no. 2, pp. 47-73, 1994.

[4] Burns A. and Wellings A., *A Structured Design Method for Hard Real-Time Systems, Real Time System*, Springer, 1994.

[5] Chidamber S. and Kemerer C., "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.

[6] Daponte P., Nigro L., and Tisato F., "Object Oriented Design of Measurement Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 41, no. 6, pp. 874-880, 1992.

[7] Douglass B., *Real time UML Lecture Slides*, Telelogix, 2010.

[8] European Space Agency, http://www.esa.int/TEC/Software_engineering_and_standardisation/TECKLAUXBQE_0.html, Last Visited 2011.

[9] Fenton N., *Shari Lawrence Pfleeger: Software Metrics, A Rigorous and Practical Approach*, Thomson Learning, 2003.

[10] Gomaa H., "A Behavioral Analysis Method for Real-Time Control Systems," *Control Engineering Practice*, vol. 1, no. 1, pp. 33-72, 1993.

[11] Harel D., "Statecharts: A Visual Formalism for Complex Systems," *Journal of Science of Computer Programming*, vol. 8, no. 3, pp. 231-274, 1987.

[12] Ishikawa Y., Tokuda H., and Mercer C., "An Object-Oriented Real-Time Programming Language," *Computer Journal*, vol. 25, no. 10, pp. 66-73, 1992.

[13] Jackson M., http://jackson-system-development.co.tv, Last Visited 2014.

[14] Jackson M., *Tools and Notions for Program Construction: An Advanced Course*, Cambridge University Press, 1982.

[15] Jalote P., *A Concise Introduction to Software Engineering*, Springer Link, 2008.

[16] Jonsson D., http://www.cse.chalmers.se**,** Last Visited 2011.

[17] Kwon B., Yang S., Kim K., and Cho J, "Real-Time System Design Tools for RT0.e (Real-Time 0bject.extended)," *in Proceeding of Software Engineering Conference*, Melbourne, pp. 376-383, 1996.

[18] Laplante P., *Real-Time Systems Design and Analysis*, Wiley IEEE press, 2004.

[19] Long G., "A UML-Based Design Methodology for Real-Time and Embedded Systems," *in Proceeding of Design Automation and Test in Europe Conference and Exhibition*, Paris, pp. 776-779, 2002.

[20] Mercer C. and Tokuda H., "The ARTS Real-Time Object Model," *in Proceeding of 11th IEEE Real Time System Symposium*, Florida, pp. 2-10, 1990.

[21] Moutaz S. and Shamala S., "Hierarchical AED Scheduling Algorithm for Real-Time Networks," *The International Arab Journal of Information Technology*, vol. 3, no. 3, pp. 219-225, 2006.

[22] Northrop L., "Let's Teach Architecting High Quality Software," *in Proceeding of IEEE 19th Conference on Software Engineering Education and Training*, Hawaii, pp. 5, 2006.

[23] Paludetto M. and Raymond S., "A Methodology based on Objects and Petri Nets for Development of Real-Time Software," *in proceeding IEEE International Conference on Systems*, Le Touquet, pp. 705-710, 1993.

[24] Rollo L., *Jackson System Development, Introduction to Software Design Methodologies*, IEEE Colloquium on, 1992.

[25] Rumbaugh J., (2011, December 23). Object Modeling Technique (OMT) [Online]. Available: http://en.wikipedia.org/

[26] Sacha K., "Transnet Approach to Requirements Specification and Prototyping," *in Proceeding of Computer Systems and Software IEEE*, Hague, pp. 220-225, 1992.

[27] SDMetrics, http://www.sdmetrics.com**,** Last Visited 2011.

[28] Selic B., Gullekson G., McGee J., and Engelberg L., "ROOM: An Object-Oriented Methodology for Developing Real-Time Systems," *in Proceeding of 5th International Workshop on Computer-Aided Software Engineering IEEE*, Montreal, pp. 230-240, 1992.

[29] Shyamasundar R. and Ramesh S., *Real Time Programming Languages, Specification and Verification*, World Scientific Publishing, 2010.

[30] Stankovic J., "Misconceptions about Real-Time Computing: a Serious Problem for Next-Generation Systems," *Computer Journal*, vol. 21, no. 10, pp. 10-19, 1988.

[31] Suonio K., "Real time: further misconceptions (or half-truths)," *Computer Journal*, vol. 27, no. 6, pp. 71-76, 1994.

[32] Wang J. and Chen H., "A Formal Technique to Analyze Real-time Systems," *in proceeding of International Computer Software and Applications Conference IEEE Computer Society*, Phoenix, pp. 180-185, 1993.

[33] Wikipedia. (2011, May 2). Real Time Operating System Examples [Online]. Available: http://en.wikipedia.org/wiki/

[34] Williams R., *Real-Time Systems Development*, Butterworth-Heinemann Publications, 2006.

[35] Ziegler J., Awad M., and Kuusela J., "Applying Object-Oriented Technology in Real-Time Systems with the OCTOPUS Method," *in Proceeding of First IEEE International Conference on Engineering of Complex Computer Systems*, Florida, pp. 306-309, 1995.

Mahmood Ahmed is PhD Student at Department of Computer Science and Engineering, University of Engineering and Technology Lahore, Pakistan. He is also working as senior scientist in a research organization. His PhD scholarship has been funded by the Higher Education Commission of Pakistan. He completed his Masters Degree in Computer Science from the Department of Computer Science and Engineering, University of Engineering and Technology Lahore, Pakistan in 2005. He published his research papers in the fields of performance evaluation of computer networks and systems and in the area related to his PhD topic i-e Design Quality Metrics for real Time Environment Applications. He also attended conference on the open source tools in his University organized by KICS (Al-Khawarizmi Institute of Computer Science.

Muhammad Shoaib is Professor at Computer Science and Engineering Department at the University of Engineering and Technology (UET) Lahore, Pakistan. He received his M.Sc. in Computer Science from Islamia University, Bahawalpur, Pakistan. He has completed his Ph.D. from the University of Engineering and Technology, Lahore, Pakistan in 2006. His Post Doc. is from Florida Atlantic University, USA, in 2009. His current research interests include Information Retrieval (IR) Systems, Information Systems, Software Engineering and Semantic Web.