12-2003

UML Extensions for Real-Time Control Systems

Qimin Gao
Dematic, gaoqimin@hotmail.com

Lyndon Brown
Western University, lbrown@eng.uwo.ca

Luiz Fernando Capretz
Western University, lcapretz@uwo.ca

42nd IEEE Conference on Decision and Control, Maui, Hawaii, pp. 5932-5938, December 2003

UML Extensions for Real-Time Control Systems

Qimin Gao L.J.Brown L.F.Capretz

Department of Electrical and Computer Engineering
The University of Western Ontario
London, Ontario, Canada N6A 5B9

February 25, 2003

Abstract

The use of object oriented techniques and methodologies for the design of real-time control systems appears to be necessary in order to deal with the increasing complexity of such systems. Recently many object-oriented methods have been used for the modeling and design of real-time control systems. We believe that an approach that integrates the advancements in both object modeling and design methods, and real-time scheduling theory is the key to successful use of object oriented technology for real-time software. Surprisingly several past approaches to integrate the two either restrict the object models, or do not allow sophisticated schedulability analysis techniques. In this paper we show how schedulability analysis can be integrated with object-oriented design. More specifically, we develop the schedulability and feasibility analysis method for the external messages that may suffer release jitter due to being dispatched by a tick driven scheduler in real-time control system, and we also develop the scheduliability method for sporadic activities, where message arrive sporadically then execute periodically for some bounded time. This method can be used to cope with timing constraints in realistic and complex real-time control systems. Using this method, a designer can quickly evaluate the impact of various implementation decisions on schedulability. In conjunction with automatic code-generation, we believe that this will greatly streamline the design and development of real-time control system software.

1. Introduction

There have been many attempts to make use of object-oriented technology for real-time software. Some of them have come from the industry real arena [3, 4, 5], while others have come from academia [6, 7, 8, 9, 10]. Many of these claims are mostly based on assumption that real-time scheduling theory can be used to perform schedulability analysis. But, traditional real-time scheduling theory results [11,12,13,14] can be directly used only when the object models are restricted to look like the tasking models employed in real-time scheduling theory, as has been done in [7, 8]. In other cases, either the claims are unsupported [4] or based on less sophisticated analysis [4, 6]. Saksena and Karvels [15] provided the first attempt to apply real-time scheduling theory to the object-oriented design by use of the state-of the art in the both fields. In their paper, they show how to integrate traditional scheduliability analysis techniques with object-oriented design models based on the assumptions that the entire external message arrives perfectly on periodic or aperiodic time interval. Martins [17] provided the first attempts to commercially implement scheduling theory for UML model design by using the technologies in [15], these integrated tools allow issues on timeliness to be addressed much earlier on in the development process.

However, some critical issues regarding real-time control systems are not well addressed by the current approaches, especially because schedulability analysis for real-time control systems has not been effectively incorporated. Although some researchers [15, 16, 17] have addressed this problems by providing code synthesis of scheduling aspects and functionality aspects models, they have mainly focused on the assumptions that all external events arrives perfectly on periodic or aperiodic without release jitter and sporadic effects. In general the real–time control systems are not the case, a message may be delayed by the polling of a tick scheduler, or perhaps awaiting the arrival of a message, and some real-time control systems have messages that behave as so-called sporadically periodic; a message arrival at some time, executes periodically for a bounded number of periods, and then re-arrives periodically for a number of times, and then does not re-arrive for a larger time. Examples of such messages are interrupt handlers for burst interrupts or certain monitoring messages in real-time control systems. Until now there is no extended method of the object-oriented design methodologies to deal with these timing constraints of real-time control systems. Thus the above analysis methods need to be improved.

In this paper, we will present an approach to incorporating schedulability analysis in a UML for Real-Time (UML-RT) model-based development process [18]. Using this approach, satisfaction of the end-to-end timing constraints of real-time control systems can be verified and the schedulability analysis results will be used for aspect-oriented code generation in the model transformation and automatic code generation. The rest of the paper is organized as follows. In section 2, we briefly review basic concepts of UML-RT. Section 3 introduces schedulability analysis based on RMA. Section 4 develops the feasibility and schedulability analysis methods for real–time control systems with jitter messages and sporadically periodic messages. In section 5, we will present schedulability results for an example system based on our method. Finally we present some concluding remarks.

2. Unified Modeling Language for Real-Time Systems

The unified modeling language (UML) [1,2] is a graphic modeling language for visualizing, specifying, constructing and documenting the artifacts of software systems. UML is a widely accepted language and it is becoming a standard for object-oriented modeling. UML has a strong set of general purpose modeling language concepts, and has been designed as an open-ended language application across different domains. UML-RT, developed by ObjectTime and Rational Rose Corporation, use UML to express the original ROOM (Real-Time Object-Oriented Modeling) concepts and their extensions.

2.1 Structure Modeling

UML-RT uses the notion of capsules to describe concurrent, active objects. Capsules are objects that communication with other capsules through interface called ports, and have each their own thread of execution. Capsules differ from other classes in that it can call operations on classes. Sending messages through public port is the only method that capsules can communicate with other capsules. Figure 1 shows an example of a systems structure for Automatic Gauge Control

Systems in the tandem cold steel mill [19], consisting of several active objects, and interconnections between objects through ports.

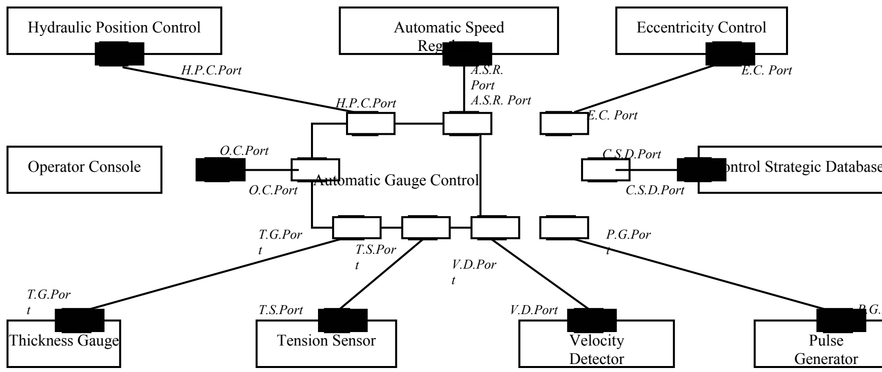

Figure 1. Object Structure Diagram for Automatic Gauge Control Systems

2.2. Behavior Modeling

In addition to the structure modeling, the capsules have their behavior defined by UML's hierarchical state machines and sequence diagrams. Sequence diagrams illustrate capsule interactions through message exchanges in a time sequence. Every capsule in the sequence diagram has a lifeline. Time progresses from top to bottom along a lifeline. The sequence diagrams use directed message arrows to describe messages sent from one capsule to another. The horizontal dimension represents the different objects in the interaction.

3. Real-time Scheduling theory

Scheduling theory for real-time systems has received a great deal of attention. The first contribution to real-time scheduling theory was made by Liu and Layland [11], they developed optimal static and dynamic priority scheduling algorithm for hard real-time sets of independent tasks. Since then, significant progresses have been made on generalizing and improving the schedulability analysis. The authors developed exact schedulability analysis to determine worst-case timing behavior for task with hard real-time constraints in the RMA model considered in the initial work [11], as well as extended models, such as arbitrary deadlines, release jitter, sporadic and periodic tasks [12, 13, 14, 20, 21, 22, 23].

Most of the deterministic schedulability analysis techniques follow the same approach. First, the notion of the critical instant of a task is defined to be an instant at which a request for that task will have the largest response time. Then, the notion of busy period at level ‘ i ’ is defined to be a continuous interval of time during which events of priority ‘ i ’ or higher are being processed [11]. With these concepts, the calculation of the worst-case response time of an action involves the computation of the response time for successive arrivals of the action, starting from a critical instant until the end of the busy period, also the response time of a particular instant of action can be calculated by considering the effects of the blocking factor from lower priority actions and the interference factor from higher or equal priority actions, including the previous instance of the same action. If the worst-case response time of the action is less than or equal to it’s deadline, the action can be said to be schedulable and feasible. Otherwise, the action is not schedulable or feasible.

4. Schedulability Analysis and Extended Sequence Diagram of UML-RT

4.1. Analysis Model

In our paper, we assume that real-time control systems are implemented in a uni-processor single thread environment, and it is made up of a set of transactions, where transaction denotes a single end-to-end computation within the system. Specifically, it refers to the entire causal set of actions executed as a result of the arrival of an external event that originated from an external source. External event sources are typically input devices (such as sensors) that interrupt the CPU-running embedded software. These external events can be periodic or aperiodic, and also have jitter and sporadically periodic characteristics. We express the real-time control system as a collection of transactions that capture all computation in the design model. We also use the term action to capture the processing information associated with an external or internal event. In our model, an action captures this entire run-to-completion processing of an event. The execution of an action may generate internal events that trigger the execution of other actions. Thus, each transaction can be expressed as a collection of actions and events. Each action is a composite action, and composed from primitive sub-actions, these primitive sub-actions include send, call, and return actions [15], which generate internal events through sending messages to other objects. We use an extended sequence diagram from UML to describe transactions in the system models. In the extended sequence diagram, we capture the detail of the processing associated with an event. Figure 2 describes the transaction of automatic gauge control system in a steel mill. The transaction is driven by a timeout message with jitter characteristics. As can be seen, the automatic gauge control object obtains the steel plate thickness from the Thickness Gauge object using a synchronous call action. It then does the control law calculations and generates a position value, which is sent asynchronously to the hydraulic position control object, the hydraulic position control object then sends a command to the hydraulic position actuators adjusting the thickness of the steel plate. The sequence diagram for a transaction can easily be extended to include sub-actions associated with code executed by the real-time execution framework.

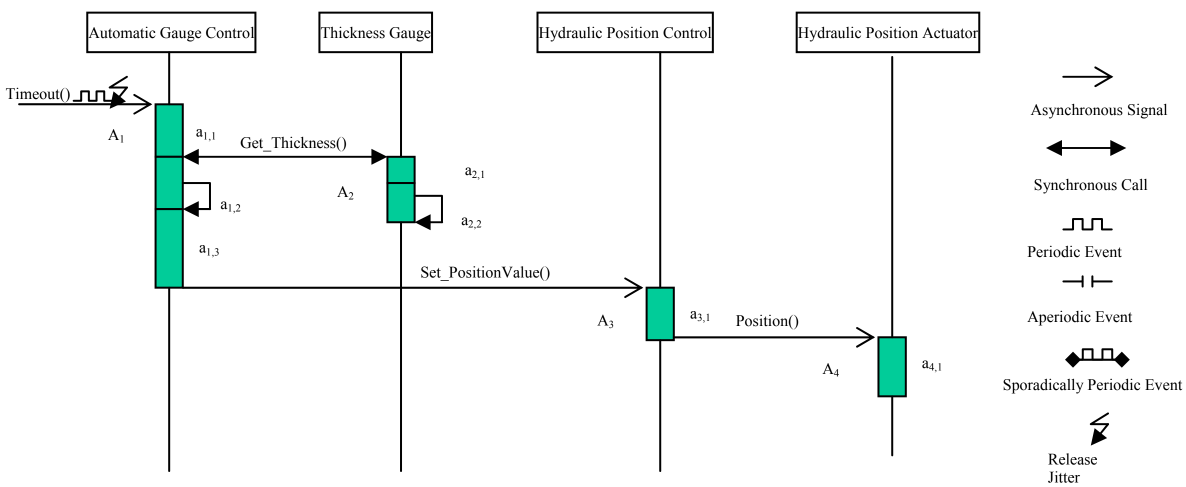

Figure 2. Extended Sequence Diagram of Automatic Gauge Control System

The extended sequence diagram can capture the timing constraints [1,2]. For the purpose of this paper, we are concerned about (1) arrival patterns of the external events, and (2) end-to-end deadlines of actions in the extended sequence diagram. The end-to-end deadlines can be specified on any action in a transaction, which is relative to the arrival of the external event.

4.2. Notation

In our paper, as defined in [15], we use event and message as synonymous. Let $\varepsilon = \{E_1, E_2, E_n, E_{n+1}, \ldots, E_N\}$ represent the set of all event-streams in the system, where $E_1, E_2, \ldots, E_n$ denote external event streams, and the remaining internal ones. All external events are assumed to be asynchronous, periodic, aperiodic events and sporadic events with release jitter. We use J_i to represent the jitter time of external event E_i. T_i and t_i represents the outer period and inner period for sporadically periodic external events E_i. If the external event without sporadic effects, the inner period of such event is equal to it's outer period. Each external event stream E_i corresponds to a transaction τ_i. We also use A_i to represent an action that associated with each event E_i. An action may be decomposed into a sequence of sub-actions $A_i = \{a_{i,1}, a_{i,2}, a_{i,3}, \ldots, a_{i,n_i}\}$, where each a_{ij} denotes a primitive action, such as sending message, calling message, and returning message. Within this model, each action A_i represents the entire "run-to-completion" processing associated with an event E_i, and it is characterized as either asynchronously triggered or synchronously triggered, depending on whether the triggering event is asynchronous or synchronous. Each action A_i executes within the context of an active object (capsule) Õ(A_i), and it is also characterized by a priority ($\pi(A_i)$), which is the same as the priority of its triggering event E_i. Each action A_i is also characterized by the computation time ($C(A_i)$) and deadline ($D(A_i)$). Each sub-action a_{ij} of A_i is characterized by a computation time $C(a_{ij})$ (abbreviated as C_{ij}); the computation time of an action is simply the sum of its component sub-actions, i.e., $C(A_i) = \sum_j C_{ij}$, also, the computation time of any sequential sub-group of sub-actions a_{ip} to a_{iq} where $p \le q$ is $C_{i,p\ldots q} = \sum_{j=p}^{j \le q} C_{ij}$. Each event and action is part of a transaction. For the rest of this paper, we use superscript to denote transactions. For example, A_i^τ represents an action and E_i^τ represents an event, both of which belong to transaction τ. Adding the superscript for external events {E_k: k=1, 2, …, n} is unnecessary since there is exactly one external event associated with each transaction, i.e., external event E_k belongs to transaction k and would be denoted as E_k^k. In this case, the superscript will be omitted.

Communication Relationships

We assumed that there are two types of communication relationships between actions, asynchronous and synchronous. We use symbol "→" to denote asynchronous relationship. An asynchronous relationship $A_i \rightarrow A_j$ indicates that action A_i generates an asynchronous signal event E_j (using a send sub-action) that triggers the execution of action A_j. Likewise, we use symbol "↔" to denote synchronous relationship. A synchronous relationship $A_i \leftrightarrow A_k$ indicates that action A_i generates a synchronous call event E_k (using a call sub-action) that triggers the execution of action A_k. We assume that if the events have a synchronous relationship, the actions have the same priority. We also use a "causes" relationship, and use the symbol $\propto$ for that purpose. The relationship captures the causal relationship between actions. Both asynchronous and synchronous relationships are also causes relationships, i.e., $A_i \rightarrow A_j$ => ($A_i \propto A_j$), and $A_i \Leftrightarrow A_j$ => ($A_i \propto A_j$), Moreover, the causes relationship is transitive, thus ($A_i \propto A_j$) $\wedge$ ($A_j \propto A_k$) => $A_i \propto A_k$. When $A_i \propto A_j$. We say that A_j is a successor of A_i since A_i must execute (at least partially) for A_j to be triggered.

Synchronous Set

For the purpose of analysis, we define the term "synchronous set of A_i". The synchronous set of A_i is a set of actions that can be built starting from action A_i and adding all actions that are called synchronously from it. The process is repeated recursively until no more actions can be added to the list. In there, we use $\Upsilon(A_i)$ denote the synchronous set of A_i and $C(\Upsilon(A_i))$ denote the cumulative execution time of all the actions in this synchronous set. We also call A_i as the root action of this synchronous set.

4.3. A Simple Example

We will use a simple example system shown in Table 1 through the rest of this paper to illustrate our ideas. The extended system sequence diagram is shown in Figure 3. The example system consists of three transactions triggered by external events E_i, one is periodic event with release jitter, one is sporadically periodic event, and the other one is aperiodic with release jitter. All the transactions are statically assigned to a single thread. For each action, we show the sub-actions a_{ij}, their computation times as well as which internal events are generated by which sub-action. Note that within each transaction we have included both synchronous (call) and asynchronous (signal) events. Furthermore, each transaction traverses multiple objects, and has multiple priorities (due to different deadlines for different parts of the transaction).

Trans τ_i	Out.P. T_i	Inn.P. t_i	Num. n_i	Jitter J_i	Event(Type) E_i	Action A_i	Priority $\pi(A_i)$	Deadline $D(A_i)$	Object $\tilde{O}(A_i)$	Sub-action a_{ij}	Comp.Time c_{ij}	Events Generated $E_i(a_{i,j})$
τ_1	60	60	1	10	E_1 (External) E_4 (Signal) E_5 (Signal) E_6 (Call)	A_1 A_4 A_5 A_6	10 6 10 10	300 800 300 280	1 4 3 4	$\{a_{1,1}, a_{1,2}, a_{1,3}\}$ $\{a_{4,1}\}$ $\{a_{5,1}, a_{5,2}, a_{5,3}, a_{5,4}\}$ $\{a_{6,1}, a_{6,2}\}$	{5, 1, 1} {5} {2,1,2,1} {4,1}	$E_4(a_{1,2})$, $E_5(a_{1,3})$ - - - $E_6(a_{5,2})$ - - -
τ_2	900	300	3		E_2 (External) E_7 (Call) E_8 (Signal) E_9 (Call)	A_2 A_7 A_8 A_9	9 9 7 9	460 400 720 450	2 5 4 6	$\{a_{2,1}, a_{2,2}, a_{2,3}, a_{2,4}, a_{2,5}\}$ $\{a_{7,1}, a_{7,2}\}$ $\{a_{8,1}\}$ $\{a_{9,1}\ a_{9,2}\}$	{1,3,1,1,4} {9,1} {10} {50,1}	$E_7(a_{2,1})$, $E_8(a_{2,3})$ $E_9(a_{2,4})$ --- - - - ---
τ_3	1000	1000	1	5	E_3 (External) E_{10} (Call) E_{11} (Signal)	A_3 A_{10} A_{11}	8 8 5	620 600 480	3 6 7	$\{a_{3,1}, a_{3,2}, a_{3,3}\}$ $\{a_{10,1}, a_{10,2}, a_{10,3}, a_{10,4}\}$ $\{a_{11,1}\}$	{4,1,5} {4,1,5,1} {250}	$E_{10}(a_{3,2})$ $E_{11}(a_{10,2})$ ---

Table 1. An Example System For Schedulability And Feasibility Analysis

In our example system, events have unique priorities (termed the priority); events can arrive at any time (i.e. want to run), but can be delayed for a variable bounded amount of time (termed the release jitter) before being placed in a priority-ordered run-queue. Periodic and aperiodic events are given worst-case inter-arrival time (termed the period); and sporadically periodic events are given the outer period and inner period, a event cannot re-arrive sooner than this time, for each arrival a event may execute a bounded amount of computation, each event is associated with the action, each action is given the worst-case execution time and deadline, This worst-case execution time value is deemed to contain the overhead due to context switching. The cost of pre-emption, within the model, is thus assumed to be zero.

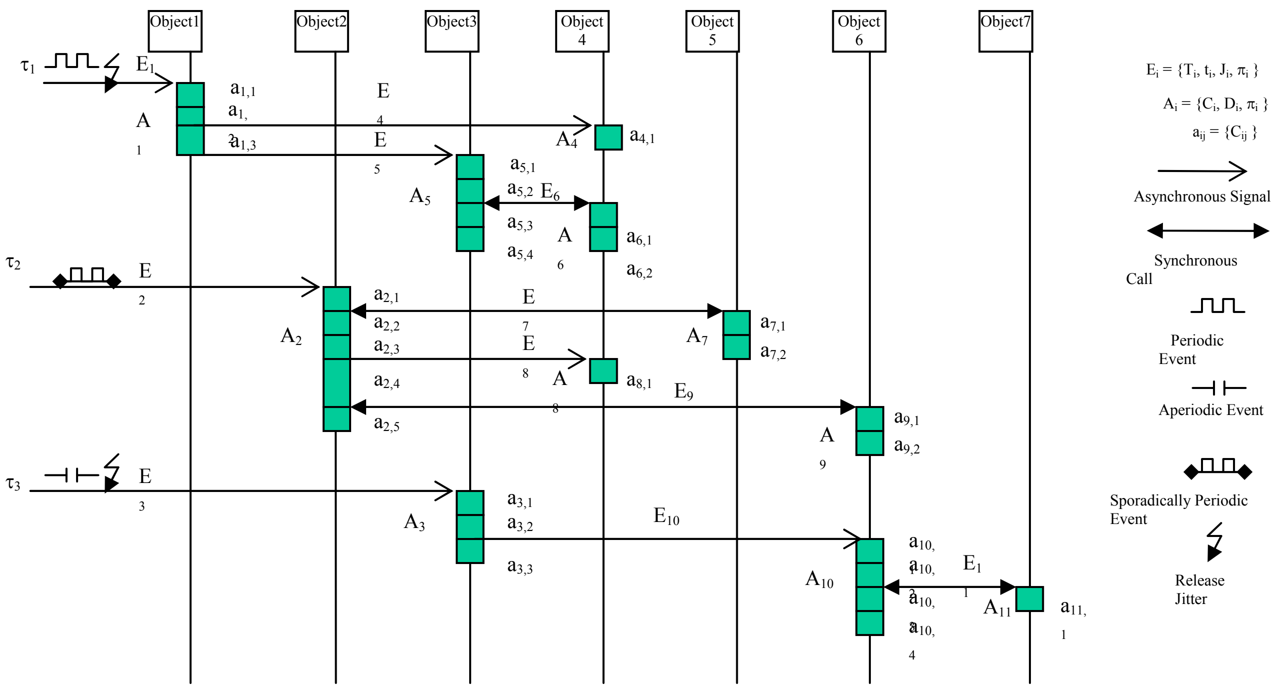

Figure 3. Extended Sequence Diagram of The Example System

4. 4. Schedulability and Feasibility Analysis

In our real-time control system model, we assume that only the external event has release jitter problem, and the internal event does not have jitter problem, because the internal event arrival is only decided by the action that represents the entire "run-to-completion" processing associated with the internal event. For the external events E_τ which behave as 'sporadically periodic' executing with an inner period (t_τ) and outer period(T_τ). we assume that the 'burst' behavior must finish before the next burst (i.e., $n_\tau t_\tau \le T_\tau$), where n_τ is the number of release of external events E_τ in a burst, and also we assumed that the release jitter (J_τ) of external event E_τ is the inner release jitter (i.e., each release of external events E_τ can suffer this jitter). In our analysis model, we carry out the schedulability and feasibility analysis by calculating the worst-case response time of actions, the worst-case response time of actions A_i^τ is calculated relative to the arrival of the external event E_τ that triggers the transaction τ . If the worst-case response time of an action is less than or equal to it's deadline, the action is schedulable, if all the worst-case times of actions in the systems are less than or equal to their deadline; the system is schedulable or feasible. We use the well-known critical instant/busy-period analysis [6, 11, 12, 14] developed for fixed priority scheduling, In our uni-processor single thread implementation environments, a priority inversion occurs if a lower priority event is processed, while a higher priority event is pending. In the same way, a level-*i* busy period is a continuous interval of time during which events of priority "*i*" or higher are being processed.

4.4.1. Worst-Case Response Time Analysis

In the worst-case response time analysis for action A_i^τ, we will compute the response time of the action for successive arrivals of the transaction, staring from a critical instant, until the end of the busy period. We let $S_i^\tau(q)$ denote the worst-case start time for instance 'q' of action A_i^τ (i.e., when the instance 'q' of the action gets the CPU for the first time), starting from the critical instant (time 0). Likewise, $F_i^\tau(q)$ denotes the worst-case finish time, starting from the critical instant (time 0). $A_{rr\tau}(q)$ denotes the arrival time of instance 'q' of external event E_τ starting from the critical instant (time 0). According to our system model, we not only consider the busy-period starting at time $J_\tau + qT_\tau$, but also consider busy-period starting at $J_\tau + q\, t_\tau$ before the release of event E_τ. In order to do that, we define two integers M_τ and m_τ, where M_τ is the number of outer periods previously in the window [0, $S_i^\tau(q)$], and m_τ is the number of inner periods. M_τ and m_τ are given by:

$$M_\tau = \left\lfloor \frac{q-1}{n_\tau} \right\rfloor$$

$$m_\tau = (q-1) - M_\tau m_\tau$$

Where q is an integer, and $q \geq 1$.

The arrival time $A_{rr\tau}(q)$ of instance 'q' of external event E_τ can be given as $A_{rr\tau}(q) = M_\tau T_\tau + m_\tau t_\tau$. Base on the traditional scheduling theory for real time systems [11,12,13,14], we can iteratively compute $S_i^\tau(q)$ and $F_i^\tau(q)$ for q=1,2,3… until we reach a $q=m$, such that $F_i^\tau(q) \leq A_{rr\tau}(m+1) - J_\tau$. Then, we let $R(A_i^\tau)$ denote the worst-case response time of action A_i^τ, and it is given by:

$$R(A_i^\tau) = \max_{q \in [1,2,\ldots,m]} \{ F_i^\tau(q) + J_\tau - Arr_\tau(q) \}$$

4.4.2. Blocking

According the scheduling theory [11,15], blocking refers to the effect of lower priority actions on the response time of an action. It may be from any transaction. Let $B(A_i^\tau)$ denote the maximum blocking time of an action A_i^τ. In uni-processor single-thread implementation environments, since scheduling is non-preemptive, priority inversion is limited to one synchronous set of actions with a lower priority root action. This action has started executing just before the transaction containing A_i^τ arrives. Thus the maximum blocking time of an action is given by:

$$\mathbf{B(A_i^\tau) = \max_{1 \leq k \leq N} \{ C(\Upsilon(A_k)) :: \pi(A_i^\tau) \geq \pi(A_k) \}}$$

4.4.3. Interference Effects and Busy Period Analysis

We known that the critical instant of an action A_i^τ occurs when all transaction arrive at the same time (we denote this as time 0), and the root action of the synchronous set of actions that contributes the maximum blocking term $B(A_i^\tau)$. Since actions are executed in a non-preemptive manner, when A_i^τ starts executing, no other action can interrupt it other than any synchronous calls that A_i^τ makes. Firstly, let the early interference function $Early_k^{A^\tau(q)}(t)$ denote the interference effect of transaction *k* prior to $S_i^\tau(q)$, assuming that $S_i^\tau(q)$=t. Likewise, let the late interference function $Late_k^{A^\tau(q)}(t)$ denote the interference effect of transaction *k* for the interval $[S_i^\tau(q), F_i^\tau(q))$, assuming that $F_i^\tau(q)$=t. Then, the value for $S_i^\tau(q)$ is given by the lowest value of $W_i^\tau(q)$, it satisfies the following equation.

$$S_i^\tau(q) = \min \mathbf{W}_i^\tau(q) :: \mathbf{W}_i^\tau(q) = \mathbf{B}(\mathbf{A}_i^\tau) + \sum_{1 \le k \le N} Early_k^{A^\tau(q)}(\mathbf{W}_i^\tau(q))$$

That is, an action (instance) will start, in the worst case, at a time $W_i^\tau(q)$ if the sum of the blocking and interference effects equals $W_i^\tau(q)$, where $W_i^\tau(q)$ is the first time instant when this become true. Note that the term $W_i^\tau(q)$ occurs on both sides of the equation, this equation can be solved by iteratively refining $W_i^\tau(q)$ using the right side of the equation, starting from an initial lower bound value $B(A_i^\tau)$ in this case, as explained in [11, 15, 21].

Once $S_i^\tau(q)$ is known, we can compute $F_i^\tau(q)$, this is done by considering the additional interference effects from higher or equal priority actions that can preempt $A_i^\tau(q)$. Because in our uni-processor single thread implementation system model, there can be no preemption effects after an action has started executing, thus we have $Late_k^{A^\tau(q)}(t)$=0. So, $F_i^\tau(q)$ can be calculated as follow:

$$F_i^{\tau}(q) = S_i^{\tau}(q) + \mathbf{C}(\Upsilon(\mathbf{A}_i^{\tau}))$$

Where $C(\Upsilon(A_i^{\tau}))$ is the cumulative execution time of all the actions in this synchronous set of A_i^{τ}.

4.4.4. Early Interference Function.

The early interference function depends on whether we are considering interference from other different transaction $K \neq \tau$, or from the same transaction. i.e., $K = \tau$.

Early Interference effects from Other Different Transactions. In this case $K \neq \tau$, for any arrival of the transaction *k* in the interval $[0, W_i^{\tau}(q)]$. We have to consider the computation times of all higher or equal priority actions making up transaction *k, a*gain, any synchronous call made recursively from these actions must also be considered, we can see that they have been already included in the calculation because of our earlier assumption that the priority of a synchronously triggered action is the same as that of the caller action. Also, interference is considered for all events arrived in the window $[0, W_i^{\tau}(q)]$. Note that we have to take the closed interval, because if a higher action becomes enabled at time $W_i^{\tau}(q)$, then $A_i^{\tau}(q)$ cannot begin executing. Now consider the computation occurring in the window $[0, W_i^{\tau}(q)]$ from higher priority sporadically periodic event E_k with release jitter J_k, if the window is larger than a number of 'bursts' of E_k then the computation time from each burst amount is $n_k C(A_k)$. For the partial 'burst' starting in the window, we can treat E_k as a simple periodic event executing with period t_k over the remaining part of the window. We let F_K represents the whole number of event E_k 'bursts' starting and finishing in the window, and it is given as follow:

$$\mathbf{F}_k = \left\lfloor \frac{J_k + W_i^\tau(q)}{T_k} \right\rfloor$$

The remaining part of the window [0, $W_i^\tau(q)$] is the length $J_K + W_i^\tau(q) - F_k T_k$, hence a bound on the number of event E_k in this remaining time is F_{kr}, and it is given by:

$$\mathbf{F}_{kr} = \left\lfloor \frac{J_K + W_i^\tau(q) - F_K T_K}{t_K} \right\rfloor + 1$$

Another bound on the number of event E_k in this remaining time is n_k, since a burst can consist of at most n_k invocations of event E_k. Therefore the least upper bound number $F_{kr\min}$ can be given by:

$$\mathbf{F}_{kr\min} = \min(n_k, \mathbf{F}_{kr})$$

So the total interference of action A_i^τ from different other transaction k is given as:

$$\mathbf{Early}_{k \neq \tau}^{A^\tau(q)}(\mathbf{W}_i^\tau(q)) = (\mathbf{F}_{kr\min} + \mathbf{F}_K\, n_k) \bullet \sum_l (c(A_l^k) :: \pi(A_l^k) \geq \pi(A_i^\tau))$$

Early Interference effects from The Same Transactions. In this case $K = \tau$, it is important to distinguish between previous instance, i.e., 1,2, …, q-1 of the transaction, and all other instances after that. Accordingly, we can write;

$$\mathbf{Early}_{\tau}^{A^\tau(q)}(W_i^\tau(q)) = \mathbf{Early}_{\tau^-}^{A^\tau(q)}(W_i^\tau(q)) + \mathbf{Early}_{\tau^+}^{A^\tau(q)}(W_i^\tau(q))$$

Where the $\text{Early}_{\tau^-}^{A^\tau(q)}(W_i^\tau(q))$ is the interference effects from the past instances (1,2,…, q-1) and $\text{Early}_{\tau^+}^{A^\tau(q)}(W_i^\tau(q))$ is the interference effects of all other instances q, q+1,… that may have arrived in [0, $S_i^\tau(q)$].

The past instances of the transaction have similar effects as other transactions, since any higher or equal priority actions of the transaction must execute prior to $A_i^\tau(q)$. Thus the

$\text{Early}_{\tau^-}^{A^\tau(q)}(W_i^\tau(q)$ can be given as:

$$\textbf{Early}_{\tau^-}^{A^\tau(q)}(\mathbf{W}_i^\tau(q)) = (M_\tau n_\tau + m_\tau) \bullet \sum_l (C(A_l^\tau) :: \pi(\mathbf{A}_l^\tau) \geq \pi(\mathbf{A}_i^\tau))$$

The interference effect of instance q onwards must not count the effect of any action A_l^τ, if $A_i^\tau \propto A_l^\tau$, since if $A_i^\tau(q)$ has not executed, any action that is caused by it could not have executed either. Furthermore, we assume that multiple instances of the same action execute in order and thus, this is true for instance $q+1$ onward as well.

If the action A_i^τ is asynchronously triggered, the $\text{Early}_{\tau^+}^{A^\tau(q)}(W_i^\tau(q))$ is given by the following equations:

First, let F_τ represent the whole number of event E_τ 'bursts' starting and finishing in the window $[0, W_i^\tau(q)]$ and is given by:

$$F_\tau = \left\lfloor \frac{W_i^\tau(q)}{T_\tau} \right\rfloor$$

The remaining part of the window $[0, W_i^\tau(q)]$ is the length $W_i^\tau(q) - F_\tau T_\tau$, hence a bound on the number of event E_τ in this remaining time is $F_{\tau r}$, and it is given by:

$$F_{\tau r} = \left\lfloor \frac{W_i^\tau(q) - F_\tau T_\tau}{t_\tau} \right\rfloor + 1$$

Another bound on the number of event E_τ in this remaining time is n_τ, since a burst can consist of at most n_τ invocations of event E_τ. Therefore the least upper bound number $F_{\tau r \min}$ can be given by:

$$\mathbf{F}_{\tau r \min} = \min(n_\tau, \mathbf{F}_{\tau r})$$

So, the $\text{Early}_{\tau^+}^{A^\tau(q)}(W_i^\tau(q))$ is given by:

$$\textbf{Early}_{\tau^+}^{A^\tau(q)}(\mathbf{W}_i^\tau(q)) = \{(\mathbf{F}_{\tau r\min} + \mathbf{F}_\tau \mathrm{n}_\tau) -$$

$$(M_\tau n_\tau + m_\tau)\} \bullet (\sum_l (C(A_l^\tau) :: \neg(A_i^\tau \propto A_l^\tau) \wedge \pi(A_l^\tau) \geq \pi(A_i^\tau)$$

According to the above analysis, for the asynchronously triggered action A_i^τ, we can find start times $S_i^\tau(q)$ as follows:

$$S_i^\tau(q) = \min \mathbf{W}_i^\tau(q) ::$$

$$\mathbf{W}_i^\tau(q) = \mathbf{B}(\mathbf{A}_i^\tau) + \sum_{1 \leq k \leq N} Early_k^{A_i^\tau(q)}(\mathbf{W}_i^\tau(q))$$

$$= \mathbf{B}(\mathbf{A}_i^\tau)$$

$$+ \sum_{\substack{k \neq \tau \\ 1 \leq k \leq N}} (F_{kr\min} + F_k n_k) \sum_l (c(A_l^K) :: \pi(\mathbf{A}_l^k) \geq \pi(\mathbf{A}_i^\tau))$$

$$+ (M_\tau n_\tau + m_\tau) \bullet \sum_l (C(A_l^\tau) :: \pi(\mathbf{A}_l^\tau) \geq \pi(\mathbf{A}_i^\tau))$$

$$+\{(\mathbf{F}_{\tau r\min} + \mathbf{F}_\tau \mathrm{n}_\tau) - (M_\tau n_\tau + m_\tau)\} \bullet (\sum_l (C(A_l^\tau) :: \neg(A_i^\tau \propto A_l^\tau) \wedge \pi(A_l^\tau) \geq \pi(A_i^\tau)$$

If the action A_i^τ is synchronously triggered, the above worst staring time $S_i^\tau(q)$ for the asynchronously triggered action A_i^τ need to be improved. Now, let's consider a synchronously triggered action A_i^τ, let A_g^τ be the asynchronously triggered action, such that A_i^τ belongs to $\Upsilon(A_g^\tau)$, i.e., the synchronous-set of A_g^τ. Then we have a chain of actions, starting from A_g^τ to A_i^τ that only execute partially in this interval, and are blocked waiting for A_i^τ to execute. Note that there must be exactly one such action A_g^τ, so there is no ambiguity. This changes the interference for instances *q*, *q*+1, … of transaction τ. For instance *q*, only a part of the synchronous set $\Upsilon(A_g^\tau)$ has executed, and this should be reflected in the equation. Rather than extend the notation to explicitly define this subset. We denote this sub-action as $a^{\tau}_{g,h}$ producing the action A^{τ}_{i}, and the computation time associated with this sub-action as $C(sub(\gamma(a^{\tau}_{g,1\ldots h})))$. For instances q+1 onwards, none of the actions in the synchronous set $\Upsilon(A^{\tau}_{g})$ can cause interference, since their previous instance (*q*) is blocked. The blocking term, interference from other transaction, and interference from previous instances (0,1,2, …,*q*-1) of the same transaction remain the same, because we assumed that $\pi(A^{\tau}_{g}) = \pi(A^{\tau}_{i})$. Based the above analysis, the worst staring time $S^{\tau}_{i}(q)$ for the synchronously triggered action A^{τ}_{i} is given as follows

$$S^{\tau}_{i}(q) = \min \mathbf{W}^{\tau}_{i}(q) ::$$

$$\mathbf{W}^{\tau}_{i}(q) = \mathbf{B}(\mathbf{A}^{\tau}_{g}) + \sum_{1\le k\le N} Early^{A^{\tau}_{i}(q)}_{k}(\mathbf{W}^{\tau}_{i}(q))$$

$$= \mathbf{B}(\mathbf{A}^{\tau}_{g})$$

$$+ \sum_{\substack{k\ne\tau \\ 1\le k\le N}} (F_{kr\min} + F_k n_k) \sum_{l} (c(A^{K}_{l}) :: \boldsymbol{\pi}(\mathbf{A}^{k}_{l}) \ge \boldsymbol{\pi}(\mathbf{A}^{\tau}_{g}))$$

$$+ (M_\tau n_\tau + m_\tau) \bullet \sum_{l} (C(A^{\tau}_{l}) :: \boldsymbol{\pi}(\mathbf{A}^{\tau}_{l}) \ge \boldsymbol{\pi}(\mathbf{A}^{\tau}_{g}))$$

$$+ C(sub(\gamma(a^{\tau}_{g,1\ldots h}))) + \sum_{l} (C(A^{\tau}_{l}) :: \neg(A^{\tau}_{g} \propto A^{\tau}_{l}) \wedge \pi(A^{\tau}_{l}) \ge \pi(A^{\tau}_{g})$$

$$+\{(\mathbf{F}_{\tau r\min} + \mathbf{F}_{\tau}\mathrm{n}_{\tau}) - (M_\tau n_\tau + m_\tau) \mathbf{-1}\} \bullet (\sum_{l} (C(A^{\tau}_{l}) :: \neg(A^{\tau}_{i} \propto A^{\tau}_{l}) \wedge \pi(A^{\tau}_{l}) \ge \pi(A^{\tau}_{g})$$

4.4.5. **Schedulability Analysis.**

From the above equations, we can calculate the value of $S^{\tau}_{i}(q)$. Once the value of $S^{\tau}_{i}(q)$ is obtained from the above equations, we can iteratively compute $S^{\tau}_{i}(q)$ and $F^{\tau}_{i}(q)$ for *q*=1,2,3 …, until we reach a q=m, such that $F_i^\tau(q) \leq$ Arrτ(m+1)- J_τ . Then, the worst-case response time of action A_i^τ is given by:

$$R(A_i^\tau) = \max_{q \in [1,2,\ldots,m]} \{F_i^\tau(q) + J_\tau - Arr_\tau(q)\}$$

If the worst-case response time $R(A_i^\tau)$ is less than or equal to it's deadline $D(A_i^\tau)$, then the action A_i^τ implementation is feasible. If the worst-case response time $R(A_i^\tau)$ is larger than the deadline $D(A_i^\tau)$, then the action A_i^τ implementation is not feasible. If all the action worst-case response times in the real-time control system are less than or equal to their deadlines, we can say that the systems implementation is feasible.

5. Scheduliability Analysis for the Example System.

Now, let us revisit our example system and apply the above scheduling analysis method to analyze the system schedulability. Table 2 shows the worst-case response time of each action

Transaction	Action	Priority	Deadline	Worst Case Response Time
τ_1	A_1	10	300	267
	A_4	6	800	763
	A_5	10	300	271
	A_6	10	280	265
τ_2	A_2	9	460	447
	A_7	9	400	386
	A_8	7	720	710
	A_9	9	450	427
τ_3	A_3	8	620	598
	A_{10}	8	600	588
	A_{11}	5	480	449

Table 2. The Worst Case Response Time for The example systems

which found by this analysis method. From the table, we can see that all the worst-case response time of actions in the system is less than their deadline constraint. So the system is schedulable and feasible. From the table we can also see that the worst case response time of all actions are large due to action A_{11} which has large execution time. Since in our system model, the implementation is in uni-processor single thread environments, it causes blocking for all the actions. Based on the table, we can see that the effect of the lower priorities of action A_4 and A_8 is also reflected in their larger worst case response time because of the greater interference. For non-preemptive scheduling in our uni-processor single thread environments, the worst case response time of the lowest priority action A_{11} is relatively lower, once the action starts executing, it executes as if its priority is raised to the highest priority in the system.

6. Conclusion

Software design has become more and more important within the real-time control system design process since functionality implementation gradually migrated from hardware to software. Consequently, several commercial tools have become available that provide an integrated development environment for real-time control systems with object-oriented techniques to facilitate the design phase. However, these tools lack the 'real-time" support required by many of these systems. Especially those with stringent timing constraints.

As a result, we proposed a methodology for the integration of schedilability analysis techniques within UML-RT techniques to support the timing requirements in real-time control system design process. The main contribution of our paper is in the development of the worst case response time analysis for object-oriented design models in which the external events suffer release jitter and have sporadically periodic characteristics, we also extent UML sequence diagram to visually describe the timing properties in real-time control systems. This results developed are also generally applicable to any modeling language using active objects, and explicit communication between objects through message passing. This method can be used to cope with timing constraints in realistic and complex real time control systems. Using this method, a designer can quickly evaluate the impact of various implementation decisions on schedulability. In conjunction with automatic code-generation, we believe that this will greatly streamline the design and development of real-time control system software.

References

[1]. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guid*e. Addison-Wesley, 1999.

[2]. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manua*l. Addison-Wesley, 1999.

[3]. B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modelin*g. John Wiley and Sons, 1994.

[4]. M. Awad, J. kuusela, and J. Ziegler. Object-Oriented Technology for Real-Time Systems: A Practical Ap-proach using OMT and Fusion. Prentice Hall, 1996.

[5]. B. P. Douglass. *Doing Hard Time: Developing Real-Time Systems with Objects, frameworks, and Pattern*s. Addison-Wesley, 1999.

[6] H. Gomaa. *Software Design Methods for Concurrent and Real-Time System*s. Addison-Wesley Publishing Company, 1993.

[7]. A. Burns and A. J. Wellings. HRT-HOOD: A Design Method for Hard Real-Time. *Real-Time System*s, 6(1):73–114, 1994.

[8]. L. Kabous and W. Nebel. Modeling hard real-time systems with uml the ooharts approach. In *Proceedings, International Conference on Unified Modeling Language (UML'99*), 1999.

[9]. K. H. Kim. Object structures for real-time systems and simulators. *IEEE Compute*r, pages 62–70, August 1997.

[10]. Yau, S.S.; Xiaoyong Zhou**;** Schedulability in model-based software development for distributed real-time systems . Object-Oriented Real-Time Dependable Systems, 2002. (WORDS 2002). Proceedings of the Seventh International Workshop on , 2002 page(s):45-52

[11]. C. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard real-time environment. *Journal of the AC*M, 20(1):46–61, January 1973.

[12]. J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and aver-age case behavior. In *Proceedings of IEEE Real-Time Systems Symposiu*m, pages 166–171. IEEE Computer Society Press, December 1989.

[13]. M. Harbour, M. Klein, and J. Lehoczky. Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority. In *Proceedings, IEEE Real-Time Systems Symposiu*m, pages 116–128, December 1991.

[14]. K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *The Journal of Real-Time System*s, 6(2):133–152, March 1994.

[15]. M. Saksena, P. Karvelas. Designing for schedulability: integrating schedulability analysis with object-oriented design. In Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on , 2000 Page(s): 101 -108

[16]. M. Saksena, P. Karvelas, and Y. Wang. Automatic synthesis of multi-tasking implementations from real-time object-oriented models. In *Proceedings, IEEE International Symposium on Object-Oriented Real-Time Distributed Computin*g, March 2000.

[17]. Paulo Martins, Integrating Real-Time UML Models with Schedulability Analysis. White Papers in Tri-Pacific Software Products 2003 http://www.tripac.com/html/whitepapers/umlsched.pdf

[18]. B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. White Paper, Published by ObjecTime, and available from www.objectime.com, March 1998.

[19]. *Tezuka, T.; Yamashita, T.; Sato, T.; Abiko, Y.; Kanai, T.; Sawada, M.* Application of a new automatic gauge control system for the tandem cold mill Industry Applications, IEEE Transactions on , Volume: 38 Issue: 2 , March-April 2002 Page(s): 553 –558.

[20]. J.Y.T.Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-times tasks. Performance Evaluation (Netherlands), 2:237-250, 1982

[21]. M.Joseph and P. Pandya. Finding response times in a real-time systems. Computer Journal (British Computer Society), 29(5):390-395, 1986

[22]. L. Sha, R. Rajkumar, and J. Lehocaky. Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on Computers, 39:1175-1185, September 1990.

[23]. J.P.Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In Proceedings of IEEE Real-Time Systems Symposium, pages 201-209. IEEE Computer Society Press, December 1990.

[24]. A. Bertossi and A. Fusiello. Rate-monotonic scheduling for hard-real-time systems. European Journal of Operational Research, pages 429-443,, June 1996

[25]. C. J. Fidge. Real-time schedulability tests for preemptive multitasking. The Journal of Real-Time Systems, Pages 61-93, 1998.