ELSEVIER

Information and Software Technology 44 (2002) 395–404

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

A framework for evaluating the effectiveness of real-time object-oriented models

Prabhaharan Kumarakulasingam, Hossein Saiedian*

Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS 66045, USA

Received 18 January 2001; revised 25 January 2002; accepted 1 February 2002

Abstract

The design of a real-time system needs to incorporate methods specifically developed to represent the temporal properties of the system under consideration. Real-time systems contain time and event driven actions. Structured design methods provided a reasonable set of abstractions for design of time and event driven factors in real-time designs. As program complexity, size, and time to market pressure grows, the real-time community migrated towards object-oriented technology. Evidence suggests that object-oriented technology in non-real-time systems is most effective in abstraction, modeling, implementation, and reuse of software systems. Many design models and methods exist for object-oriented real-time designs. However, the selection process of a model for a particular application remains a tedious task. This paper introduces an analysis framework that can be applied to a design model to evaluate its effectiveness according to desired performance specifications. To illustrate our approach, we present a case study using the popular automotive cruise control example on two real-time object-oriented models. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Hard real-time hierarchical object-oriented design; Real-time object-oriented models; Unified modeling language; Unified modeling language real-time

1. Introduction

Rapid advances in computing, electronics and communications necessitate the implementation of software driven embedded computerized control in many real-time systems. Real-time systems control the environment by reading sensory data from the environment, performing computations on the data and then reacting to the results of the computation. The correctness of the system depends on both the logical and temporal response. To ensure predictable real-time behavior under abnormal operating conditions, real-time software needs to be both dynamic and reliable.

According to Saksena [1], real-time systems contain time and event driven software. Time driven software is driven by periodic time-triggers in the system. Therefore each task is modeled as a periodic task and contains a deadline for completing its computation. Event driven software is asynchronous and inherently non-deterministic. The system responds to real world events as and when they occur and its future behavior is determined by the current state of the system.

The temporal aspect of real-time systems makes real-time design inherently different from other system designs. Temporal behavior of a system depends on the environment that the system interacts with. Furthermore temporal behavior requirements are usually stated for a chain of tasks that constitute a specific function [2]. Correct system behavior imposes constraints related to time in the real world. Therefore real-time requirements need to be specified at the beginning of an analysis workflow and these requirements need to be periodically validated throughout the development process. The extents to which they are specified and modeled drive the success or failure of these systems. Therefore early modeling and analysis aid the success of a system.

Real-time requirements can be subdivided into three groups. They are behavioral requirements, temporal requirements and cost requirements [2].[1] A real-time model is composed of tasks that describe the behavior of the system, and resources that are used by these tasks. System behavior restricts implementation, which affects scheduling constraints. Several techniques have been used to develop real-time systems. They include Finite State Machines, Petri

* Corresponding author. Tel.: +1-913-897-8515; fax: +1-913-897-8490.
E-mail address: saiedian@eecs.ukans.edu (H. Saiedian).

[1] Cost and behavioral requirements are beyond the scope of this article and thus are not discussed further.

0950-5849/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0950-5849(02)00023-X

Nets and Timed CSP, and more recently the Unified Modeling Language Real-Time (UML-RT) [3]. Real-time systems are analyzed by ensuring that each task in a system is schedulable and will complete within the specified time limits. Rate Monotonic Analysis (RMA) is one method of analyzing real-time systems for schedulability. RMA uses an analytical approach to verify real-time constraints of a system by considering the ratio of task computation time to task period time for a set of n tasks. RMA predicts whether a system is schedulable within a given set of requirements, and analyses if tasks are schedulable under all conditions.

Object-oriented programming is an improvement over procedural programming, particularly in the areas of code reusability, adaptability and understandability. Furthermore, object-oriented systems represent the natural world directly in a software program [4]. These type of systems promote a better understanding of the problem space by using abstractions like active objects [5]. However, current object-oriented real-time models are weak on specifying temporal, behavioral, and dependability requirements [6]. Real-time object-oriented modeling requires to represent concurrency and synchronization among processes. Also the real-time constraints of system functionality need to be expressed explicitly and unambiguously [7]. There have been many attempts both in industry and in academia [3,5] to apply object-oriented concepts to real-time systems.

A number of solutions contained visual notations and modeling abstractions to aid a designer understand the problem space and possible effects of these solutions at an early stage of a development cycle. However, a weakness of these systems has been the lack of support for proving a system schedulable [5]. One can choose from a set of tools that aid in an object-oriented real-time design process. Some of the popular tools include: ObjecTime Developer, Real-Time Rational Rose, and Rhapsody. These tools provide a framework for representing and analyzing a design solution to the problem space. Although some of these tools extend the modeling language to support real-time systems, they provide little or no facility for schedulability analysis. As we discussed earlier, a schedulability analysis provides conclusive evidence that a real-time system meets its functional timing constraints. Various modeling languages [8] provide support for object-oriented real-time design. Models like Hard Real-Time Hierarchical Object-Oriented Design (HRT-HOOD) [9], Real-Time Object-Oriented Modeling (ROOM) [10], and UML-RT [11] are object-oriented models that help to analyze the temporal real-time properties of a system.

The design models and languages available to an object-oriented real-time designer today vary in their representational and analysis strengths. The strengths and weaknesses of these methods are not easily discerned during the model selection process. Designers new to object-oriented real-time design may extend an enormous amount of effort discerning the strengths and weaknesses of these models for a particular application. In this paper, we address this problem by illustrating a case study that can help with evaluating the essentials of an object-oriented real-time design model. We state a set of evaluation criteria and provide evidence as to why these criteria are important and also provide guidelines for discerning whether a model is suitable to represent and analyze a real-time system.

Section 2 provides a brief analysis of various methods available in the market for real-time systems modeling and the how well they address the issue of schedulability. In Section 3 we introduce the problem domain and our analysis framework. In Section 4, we analyze two models HRT-HOOD and ROOM that seem equally effective to a designer who is well versed in the area of object-oriented design. To illustrate our approach we present a case study using the cruise control example given by Booch [12]. We use the following criteria for analysis purposes:

Criteria 1: *Semantics*. To examine the semantics of object-oriented models for annotating and analyzing real-time requirements.
Criteria 2: *Task priority and response time requirements*. To examine the models' ability to illustrate the notion of task priorities for satisfying functional correctness and non-functional response time requirements.
Criteria 3: *Priority assignment*. To evaluate the effectiveness of priority assignment mechanisms.

Section 5 presents a discussion of our findings.

2. Object-oriented real-time modeling methods

In recent years, many designs and modeling tools have been proposed for modeling real-time systems to depict its real-time constraints. Only a few of these models address real-time object-oriented modeling. In this section, we discuss the suitability of the following models for representing real-time objects and their corresponding evaluation methods. We have studied the following: TMO [6], OPM/T [7] RTSO-RAC [13], ROOM [10], and HRT-HOOD. A real-time system is considered to meet its timing constraints if it can be proven to be deterministic and schedulable. Schedulability analysis is performed using the RMA equation. In this section, we evaluate the essential real-time representations of the models and discuss their evaluation framework.

Cingiser and Ma [13] describe the RTSO-RAC model where objects contain real-time attributes, methods, constraints and compatibility functions. Each attribute includes its age, temporal consistency constraints, and imprecision. Methods are composed of a set of arguments, sequence of operations, execution times and operational constraints. Operations in methods are further described by dead lines and worst case execution times. This allows the operations to be analyzed for schedulability and

concurrency. The compatibility function provides a mechanism to express method concurrency in objects. Cingiser and Ma [13] do not describe the schedulability or the timeliness aspect of their model. They state that the constraints, real-time attributes and the compatibility function impose a higher degree of concurrency control in objects that allows the system to potentially meet its timing constraints.

Peleg and Dori [7] describe the OPM/T system as a model where objects and processes are considered as entities. Temporal, trigger and constraints are specified on object methods. Each object also transits through many states during the application. When objects cannot respond to messages, they raise an exception and the object method is called again. Objects may not be able to respond to messages because they are busy executing another method or they may be in the wrong state in which case the receiving object does not receive the message. Although OPM/T allows for the specification of real-time parameters, the evaluation method does not explore the effects of concurrency, blocking or priority assignments.

Kim [6] describes real-time objects as time-triggered and message triggered objects (TMO). TMO consists of object data stores and service methods. The data stores are composed of lockable segments each containing data members. The service methods include spontaneous methods, which are only called from hardware interrupts such as a timer. All other object interaction is carried out through a conventional message passing mechanism. A server object guarantees timeliness by relying on the operating system and the underlying hardware to execute the service methods within its timing constraints. Furthermore, spontaneous methods are assigned a higher priority. The success of this model depends on the designer's knowledge of the operating system and the underlying hardware platform. This poses difficulty when the model is evaluated for its effectiveness in representing real-time systems.

The CHAOS system described by Bihari and Gopinath [4] is an object based programming, execution and language model. Objects in this model consist of attributes such as static and dynamic execution times and degree of concurrency. Concurrent objects also provide a coordinator object that ensures mutual exclusion on shared data structures. Method execution times are derived from worst case experimental data and these results are used for temporal analysis. Messages offer a combination of functionality and execution over-head, supported by a mechanism to synthesize application specific messages. Designers can then choose or synthesize the appropriate message type for their application environments. This results in these messages being statically generated, which ensures that system-timing constraints will be met. Bihari and Gopinath [4] do not specify an analytical technique for guaranteeing timeliness aspects of a real-time system but rather leave this to the self imposed behavior of each collaborating object, where each object manages and negotiates the timeliness criteria before sending messages for execution to other objects. This method seems reasonable for soft real-time systems but is difficult to justify for hard real-time systems, as an object method invocation cannot be postponed to another time interval.

Saksena and Karvelas [14] have applied real-time schedulability analysis to object-oriented real-time systems. Schedulability analysis is carried out by computing response times for each action and by applying these times to an RMA equation. The response time is the sum of all individual object execution times that participate in the action. Therefore an RMA analysis can determine an object-oriented real-time systems performance. During our review we found that HRT-HOOD and ROOM models were closely comparable to each other and supported comparable mechanisms for representing and evaluating real-time systems. Therefore we have chosen these two models to apply our evaluation framework theory. We describe and analyze the models in Sections 3 and 4.

3. Research methodology

3.1. Analysis framework

According to Ishikawa and Tokuda [15], the principle of schedulability governs real-time systems analysis. When a system is deterministic and schedulable to meet its timing constraints it is considered correct. Two major approaches used for developing schedulable systems include the cyclic executive and rate monotonic scheduling. Cyclic executives schedule events based on a major time period that is composed of many minor time cycles. Moreover the major time cycle schedules the minor cycles to be repeated indefinitely. Minor cycles are composed of tasks. Timing constraints are then achieved by fitting each task within a minor cycle. The second approach, rate monotonic scheduling, analyzes and quantifies task execution periods and priorities within a given constraint. This approach assigns priorities to tasks and analyzes the processor utilization of a task and its period of execution. Tasks with shorter periods are thus assigned higher priorities. The system is then analyzed for the following constraint for n tasks as given by Liu [16]:

$$\sum_{i=1}^{n} \frac{C(i)}{T(i)} n(2^{1/n} - 1)i = 1 \quad (1)$$

where $C(i)$ is the CPU utilization time and $T(i)$ is the period of the task.

According to Liu [16], task priorities in process oriented real-time systems are used to effectively schedule the system to meet its timing constraints. A high priority task preempts the execution of a lower priority task by completing its own execution and then resuming execution of the lower priority task. Since data structures can be shared between tasks, shared data needs to be protected. This

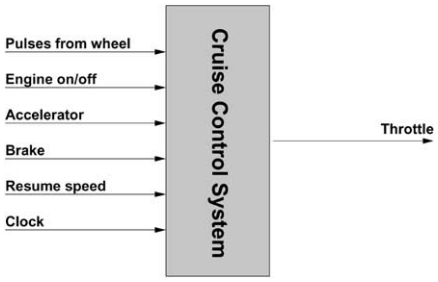

Fig. 1. Block diagram for cruise control system [12].

protection is accomplished by providing guard mechanisms called semaphores. The application of a semaphore results in a lower priority task preventing the execution of a higher priority task since the shared data is locked and in use by the lower priority task. This situation is called blocking and can be overcome by assigning task priorities dynamically [16]. Therefore, in order to support a real-time design an object-oriented model needs to be able to represent and simulate the execution period, deadline and dynamic priorities of each object. Our analysis consists of the following:

1. Evaluating a candidate models object structures for real-time attribute notations.
2. Evaluating mechanisms available for depicting, implementing and analyzing task priorities for satisfying response time requirements.
3. Evaluating whether a model supports provisions for dynamic priority assignments.

We apply the above criteria to the HRT-HOOD and ROOM models to illustrate our evaluation framework. The models are subsequently studied for their effectiveness in representing and analyzing a real-time system for schedulability using RMA. The effectiveness of the model with respect to real-time constraints is then discussed. Finally, we show the reader that a generic methodology can be effectively used for choosing a particular object-oriented real-time model.

3.2. Problem scope and definition

The cruise control model used by Saksena et al. [17] is used in this study. Fig. 1 displays the block diagram of the system as defined by Booch [12]. The purpose of the control system is to maintain the speed of a vehicle over varying terrain in cruise mode. When the brake is applied, the system decreases speed and transfers control of the vehicle to the driver. Three transactions are used in the study:

1. Closed loop feedback control
2. Entering automatic control
3. Exiting automatic control

Closed loop feedback control is triggered by a periodic timer output. Its primary function is to maintain the speed of the vehicle. Cruise mode is exited by pressing the brake, switching the cruise lever off, or pressing the accelerator pedal. Switching the cruise lever on and releasing the accelerator pedal places the system back in cruise mode.

Fig. 2 shows the state transitions of the cruise control object as depicted in our study. First the system is initialized to the manual control state. When cruise is enabled the system transitions to the automatic control state. A brake pressed event in the automatic control state reverts control back to manual mode. Table 1 below provides the cruise control transactions and their timing constraints. Table 2 shows the computational times used for tasks.

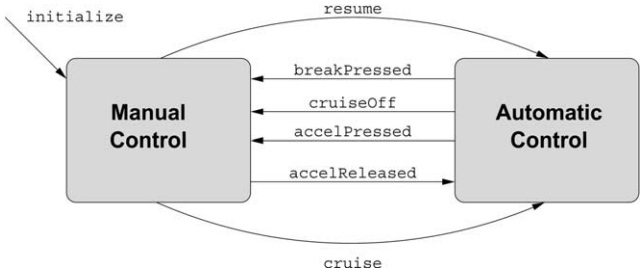

Fig. 2. Cruise control state diagram [17].

4. Analysis

4.1. Elements of the HRT-HOOD model

The HOOD method combines the advantages of object-oriented methods and hierarchical decomposition and control structuring. The objects interface and the internal representation implement the functionality of the object, while the control structure provides the calling mechanisms necessary for invoking other objects. HRT-HOOD extends this basic methods to provide support for designing and analyzing real-time software systems. The design cycle is divided into logical and physical design activities. Logical design activities implement the functional requirements of the system and physical design activities implement non-functional requirements. Physical design activity specifically aides schedulability analysis that ensures the temporal correctness of a system [18].

HRT-HOOD provides real-time modeling support by depicting each object's attributes with real-time abstractions. The logical design activity produces five different types of real-time objects that describe the functionality and the physical design activity produces the timing requirements of the system. Synchronous, asynchronous, and protected access constraints are also placed on an object's behavior.

HRT-HOOD includes PASSIVE, ACTIVE, PROTECTED, CYCLIC and SPORADIC objects. [18] describe the objects as presented below:

PASSIVE objects do not spontaneously invoke operations

Table 1
Cruise control transactions and their timing constraints [17]

Transaction	Stimulus	Response	Period (ms)	Deadline (ms)
Shaft interrupt (SI)	Shaft interrupt	–	10	10
Determine speed (DS)	Timeout	Speed value	50	10
Control loop (CL)	Timeout	Throttle value	100	100
Brake pressed (BP)	Brake pressed	–	–	50
Enter cruise (EC)	Resume	Throttle value	–	200

in other objects and have no control over the invocation of their own operations. This means that whenever a passive object is invoked control is immediately transfered to that object. Also all passive objects contain sequential code that do not synchronize with any other object.

ACTIVE objects can control when their methods can be invoked and may spontaneously invoke operations in other objects. ACTIVE objects can also contain constrained objects that are executed under the control of the object broker control system (OBCS). OBCS provides synchronization between objects.

PROTECTED objects control when their methods are invoked and do not spontaneously invoke methods in other objects. They perform the well known monitor functions in providing access to data under mutual exclusion. Therefore shared data structures can be designed as PROTECTED objects.

CYCLIC objects represent periodic activities. They spontaneously call methods in other objects, yet are limited to operations that must be executed on a periodic basis.

SPORADIC objects have their own threads of control. A SPORADIC object does not block the caller and contain their own independent threads of control. All operations that invoke a SPORADIC object shall have a minimum and maximum inter-arrival time. In addition these objects are also used to transfer asynchronous control. Finally SPORADIC objects also contain an OBCS.

In summary, CYCLIC objects provide illustration and control of time driven aspects of the system whereas SPORADIC objects support event driven aspects and support program control transfers. PROTECTED objects are used for designing controlled access to shared data structures through mutual exclusion and ACTIVE objects are used for background tasks. Furthermore, non-functional requirements of a system are encoded in an object's attributes. CYCLIC and SPORADIC objects contain temporal attributes that describe real-time properties such as period of execution, deadline, minimum arrival time, worse case execution time and priority levels. Also CYCLIC and SPORADIC objects are decomposed into terminal objects that have precedence constraints. A precedence constraint allows any activity within the terminal object to begin as soon as its predecessor has terminated.

Table 2
Computation times for tasks

Cruise control (ms)	Speedometer (ms)	Other (ms)
$C_{\text{timeout}} = 2$	$C_{\text{shaft}} = 2$	$C_{*} = 2$
$C_{\text{speedvalue}} = 10$	$C_{\text{timeout}} = 3$	
$C_{*} = 5$	$C_{\text{speedRequest}} = 3$	

4.2. Elements of the ROOM model

The ROOM model is composed of a set of objects encapsulating data and behavior and represented as state machines [17]. The objects in the system communicate with each other through message passing mechanisms. Messages are passed through interface objects. Each message contains a name, data and priority attributes. The behaviors of objects are represented as state machines. These state machines respond to event and time driven stimulus in a real-time system. Each object executes one transition at a time and run-to-completion. This implies that processing a message cannot be preempted by another message to the same object [17].

The physical architecture of the ROOM model is composed of three layers. The bottom layer contains a virtual machine that executes the compiled ROOM model. This virtual machine also manages communications among actors and timing services necessary for the model. Actors' exchange messages through port objects that are instantiations of a protocol class. Messages received by an actor are queued and processed according to the priority protocols established in the system.

4.3. Evaluation of models

4.3.1. Criteria: semantics

Objective of Criteria 1. Examining the availability of semantics in object-oriented models to notate and analyze real-time requirements.

A real-time system is composed of software components that react to periodic and asynchronous events. Therefore any real-time modeling language should be able to provide semantics for representing objects that support processing of periodic and asynchronous events. Functional requirements of a real-time system are satisfied by the contiguous execution of a set of tasks. Objects participate in the execution of a task. Therefore a real-time representative model should provide support for aggregating objects into a set of tasks.

Listing 1
Case study system in HRT-HOOD

```
1   Object Name:              Speedometer
    Object Type:              SPORADIC
    Decomposed into:          ACTIVE and PROTECTED
                              object
    a. Object Name:           Shaft Interrupt
      Object Type:            SPORADIC
      Attributes:             sensor data
                              thread priority = 1
                              periodicity = 10 ms
                              deadline = 10 ms
      Operations:             Read_shaft_data
    b. Object Name:           DetermineSpeed
      Object Type:            PROTECTED
      Attributes:             sensor data
                              current speed
                              thread priority = 2
                              periodicity = 50 ms
                              deadline = 10 ms
      Operations:             Compute_speed

2   Object Name:              Cruise Control
    Object Type:              CYCLIC
    Attributes:               current speed
                              thread priority = 4
                              periodicity = 100 ms
                              deadline = 100 ms
    Operations:               Get_current_speed
                              Update_throttle

3   Object Name:              Brake
    Object Type:              SPORADIC
    Attributes:               thread priority = 3
                              deadline = 50 ms
    Operations:               Transfer_control

4   Object Name:              Lever
    Object Type:              SPORADIC
    Attributes:               speed value
                              thread priority = 5
                              deadline = 200 ms
```

We refer to a collection of tasks as a thread in the rest of the paper. Since real-time systems need to satisfy timing constraints on their functionality, the model also should support transfer of control between threads to satisfy the time constraints of a system. We compare the representation of our case study in HRT-HOOD and ROOM below.

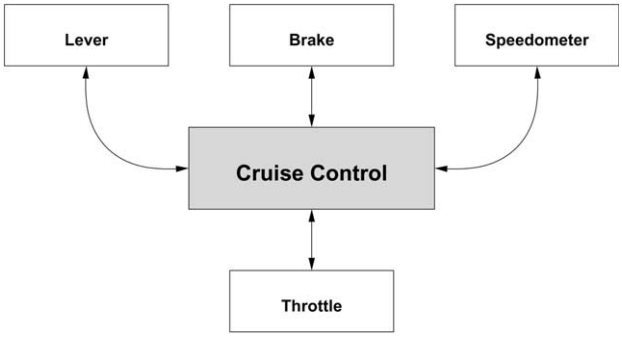

Fig. 3. Cruise control system ROOM model [19]

The logical design classifies the system under study with four objects: *cruise control*, *lever*, *speedometer* and *brake*. The object structure for HRT-HOOD is described by an object name, object type and object attributes. Where object type is defined as either CYCLIC, SPORADIC, ACTIVE, PASSIVE or PROTECTED. HRT-HOOD object attributes considered in our study include a deadline period and execution period for CYCLIC and SPORADIC objects, minimum arrival time for SPORADIC objects, thread priority and precedence constraints on threads. Our case study system under evaluation is defined in HRT-HOOD in the algorithm Listing 1. The deadline and periodic time constraints for the transactions in this study are the same as used by Saksena et al. [17] and are shown in Table 1.

All objects in the system have been characterized into the types available along with their relevant real-time attributes necessary for the study. As this study is concerned with the temporal aspect of a system, only attributes that affect the temporal execution times are considered. The speedometer object has been decomposed into SPORADIC and PROTECTED objects. This Shaft Interrupt is considered as an interrupt based task that occurs at a minimum periodic time of 10 ms. This interrupts execution time is 2 ms as read from Table 2. The Determine Speed object performs a protected read on the shaft data and computes the current speed of the vehicle. Its computation time for the task is 3 ms. A call to the Determine Speed object is also invoked by the OS timing services every 50 ms. The cruise control object is also invoked by the system's timing service every 100 ms for maintaining the control loop. Its primary function is to obtain the current speed of the device and update the throttle speed value. The brake pedal object is also checked every 50 ms by timing request service from the OS.

Each object is provided a thread priority. HRT-HOOD imposes a preemptive mode of behavior on objects. Higher priority objects can preempt lower priority objects. Therefore an operation on a lower priority thread can be preempted to run the higher priority thread. Access to shared data is protected through the use of a PROTECTED object type therefore mutual exclusion is enforced.

ROOM models the system in terms of design objects called actors. Therefore the cruise control, speedometer, brake and lever are termed actors. Each actor contains a state machine and a message queue. Messages are queued in the order of arrival but can be prioritized so that higher priority messages are at the head of the queue. This priority scheme is termed message priority. Several objects can be composed into a collection of tasks or a thread. Threads are also assigned priorities. Within each thread, processing of tasks are implemented using message priorities. Thread priorities are implemented by the operating system [17]. Actor objects fulfill the functionality of the system by a sequence of interactions with each other. Interactions are accomplished by message passing, which trigger

Table 3
Cruise control transactions and their timing constraints [17]

Transaction	Period (ms)	Deadline (ms)	Priority
Shaft interrupt (SI)	10	10	1
Determine speed (DS)	50	10	2
Control loop (CL)	100	100	4
Brake pressed (BP)	–	50	3
Enter cruise (EC)	–	200	5

transactions in actor state machines [19] that implement the functionality requested.

Fig. 3 displays the cruise control system actor structures as described by Gaudreau and Freedman [19]. We map the four actors into two independent threads as discussed by Saksena et al., [17]. Thread one contains only the speedometer object. Thread two contains the other three objects. Thread one is assigned a higher priority than thread two. Table 3 shows the transaction priorities and their deadlines and periods.

4.3.2. Criteria: task priority and response time requirements

Objective of criteria 2. Examining the models ability to illustrate the notion of task priorities to satisfy functional correctness and non-functional response time requirements.

Our analysis case would be entering the automatic control. Setting the lever to the cruise on position results in the system entering the automatic control state. In the ROOM model, this state results in a message being passed to the cruise control object from the lever object. The cruise object then requests the current speed from the speedometer object. This entails reading the latest speed data computed by the speedometer and updating the throttle. The deadline time for this transaction is 200 ms. A shaft interrupt and a determine speed request are invoked every 10 and 50 ms, respectively in the system by the ROOM virtual machine. Therefore processing of the speed request message is held until the shaft interrupt update message and the determine speed events are processed. ROOM models impose run-to-completion semantics only within the same thread. Note that the `shaftinterrupt` and determine speed transactions are service methods offered by the speedometer object and that this object resides in a different thread than the cruise control or lever object. Since the speedometer object is contained in a different thread the shaft interrupt or the determine speed interrupt events are not held until the previous thread completes. However, since the shaft and determine speed interrupt related methods are in the same thread they may block each other due to ROOM models run-to-completion semantics. This additional blocking time should be considered in the analysis. According to Saksena et al. [17] the total response time for entering automatic control is 43 ms, well within the dead line required. The collaborating objects and their message passing sequence is shown in Figs. 4 and 5.

HRT-HOOD implements the above transaction by providing synchronization and collaboration between the lever, cruise control and speedometer objects. In the HRT-HOOD model the Shaft Interrupt and Determine speed have thread priorities of one and two, respectively. Thus the Shaft Interrupt service method is always acknowledged and never blocked. Also the speedometer object contains a protected

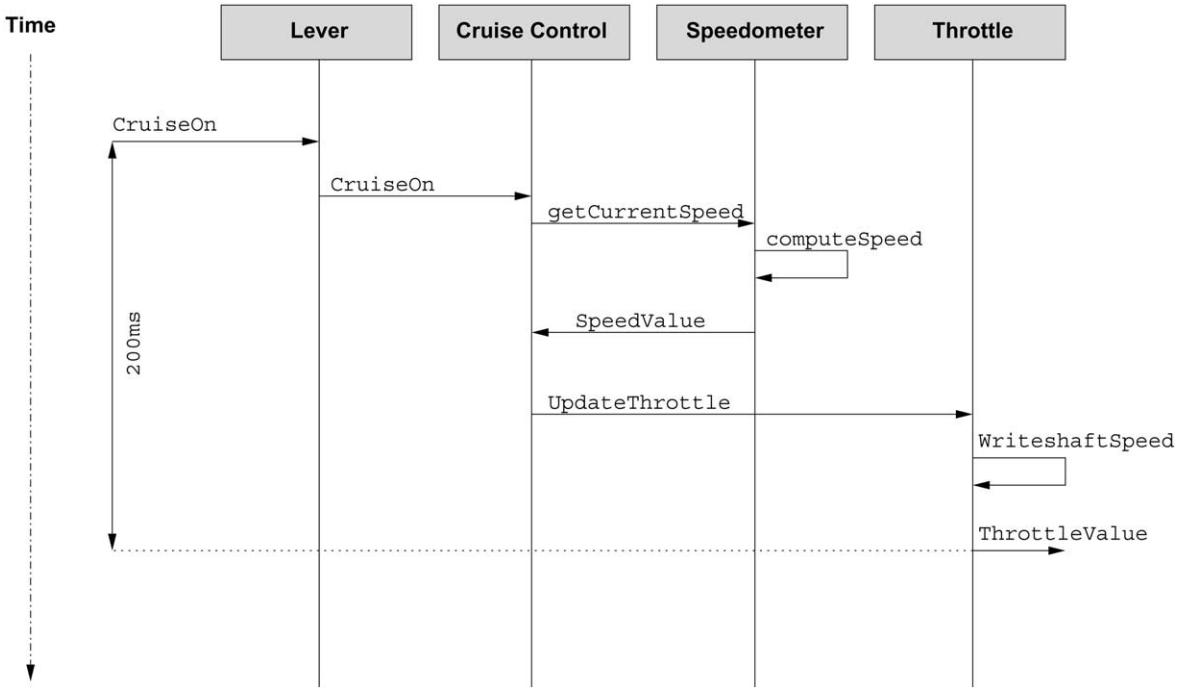

Fig. 4. Entering cruise control in HRT-HOOD.

object (determine speed) that inherently provides mechanisms due to its structure for mutual exclusion on the speed data. Therefore the chance of receiving corrupted data is eliminated. In the ROOM model, mutual exclusion have to be explicitly implemented adding additional delays during execution.

In the present transaction, the data transferred can be from a previous update of the shaft and would not adversely affect the application. However, the same cannot be said for all applications. Therefore if a model can support mechanisms for priority assignments of individual objects, and the assignment of a collection of tasks to a threads with preemptive thread processing, then that model is suitable for proving a real-time systems temporal performance.

4.3.3. Criteria: priority assignment

Objective of criteria 3. Evaluate the effectiveness of priority assignment mechanisms.

Our analysis case would exist in automatic control. Automatic control is exited by pressing the brake. The message sequence for the brake pressed transaction is shown in Fig. 6. The brake event is assigned a transaction priority of three in our example. Two other transactions, the control loop and enter cruise transactions have lower priorities than the brake transaction. ROOM responds to the event by capturing it as an interrupt and allowing its virtual machine to send a message to the brake object. The brake object in turn sends a message to the cruise object requesting a transition to manual state. If the cruise control object is in its ready state then it responds to the brake pressed message immediately by reducing the speed of the throttle. However, if the cruise control object is executing a feedback control message then it is blocked due to the run-to-completion paradigms enforced in the ROOM model as the brake and cruise objects are mapped to the same threads. Therefore brake pressed transaction runs-to-completion after the control loop transaction. The brake pressed transaction is delayed by 10 ms. Therefore the total execution time for the brake pressed transaction is 7 ms for execution of the transaction and 10 ms for blocking time of lower priority tasks. The response time deadline of a brake pressed message is 50 ms. Therefore in this instance the brake pressed transaction is able to complete within the dead line of 50 ms.

HRT-HOOD implements the above scenario as a message request from a SPORADIC brake object to the CYCLIC cruise control object. Since we are executing on a single processor, a message waits on a higher priority thread yet preempts the execution of a lower priority thread. The brake object has a priority of three. If the cruise object is already involved in a transaction to update the vehicle speed it will be preempted by the brake pressed message immediately. The discussion here indicates that the ability to assign priorities to threads versus objects is important in assessing a model for real-time schedulability constraints.

The assignment of a priority to a transactions versus objects is especially important when priorities are dynamically changed in the system. We extend our scenario by stating that the speedometer object and the brake pressed object share data and/or a control mechanism. Since the brake pressed object's execution time is longer than the speedometer object's `Read_shaft_data` and `Compute_speed` tasks, the brake pressed execution is preempted in the HRT-HOOD model. The shared data or control is changed and control returned to the brake pressed object. This results in the brake pressed object using a different set of data than it had previously used before preemption occurred. One way to address this deficiency is to change the priority level of the brake pressed transaction to be higher than the speed computation transaction. This change is easily accomplished in the HRT-HOOD model as one can change the priority attribute of the participating object. A change in the ROOM model would require the introduction of a new thread containing the brake object.

Priorities can also be changed dynamically. The HRT-HOOD models object attributes allow dynamic changes to be managed easier than ROOMS thread based architecture. Since a collection of tasks are included in a thread thread priorities can be changed dynamically. Therefore it is essential that the collection of tasks into threads be done effectively to be able to satisfy dynamic priority assignments. Overall the HRT-HOOD structure is more flexible when considering dynamic priority assignments in an application.

Once a system's objects, their behaviors and execution times of threads and tasks have been identified, a rate monotonic scheduling analysis is performed. Rate monotonic scheduling was realized for task based systems. It has been recently extended to object-oriented systems by Ishikawa and Tokuda [15]. In order to perform this analysis for object-oriented systems, the collaborating objects need to be grouped into independent threads. Eq. 1 states the RMA analysis, where $C(i)$ is the CPU utilization time, $T(i)$ the period of the task and n the number of tasks. When blocking times are considered an additional term $B(i)/T(i)$ is added to the equation as discussed by Gaudreau and Freedman [19] where $B(i)$ is the blocking time experienced by task i. If a system meets the constraints of Eq. (1), then it is said to be schedulable and meets all its real-time constraints. Therefore, if a systems tasks execution times, task periods (or minimum arrival time in case of event driven behaviors) and blocking times become known then it can be analyzed for schedulability which guarantees the hard real-time constraints of the system.

5. Discussion

This study illustrates the application of an evaluation framework for evaluating a real-time object-oriented design model. While object-oriented based design and analysis is well understood, real-time object-oriented design models are not as familiar to today's real-time practitioners. The

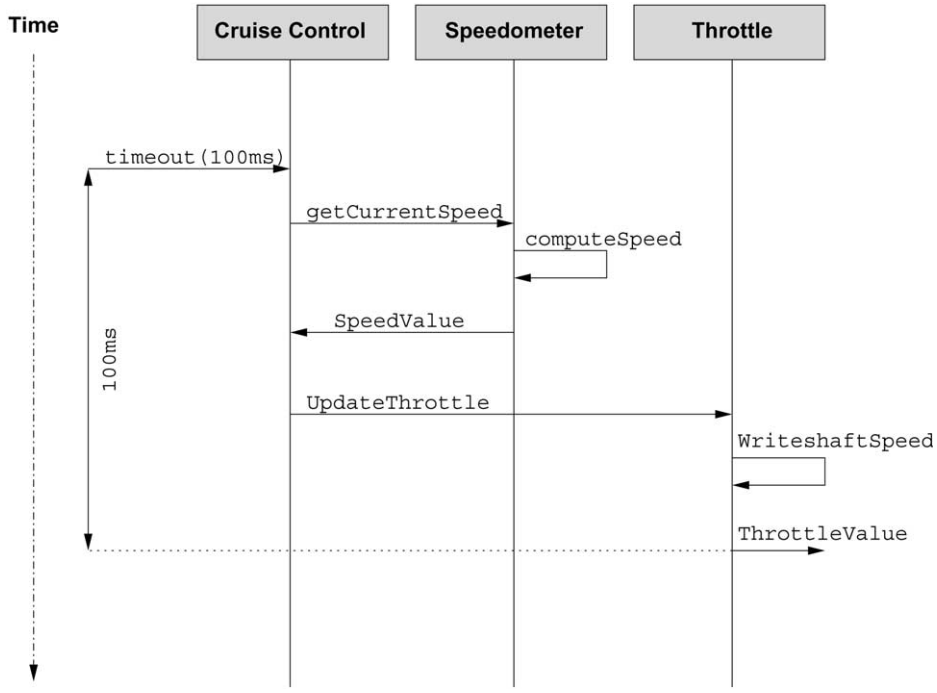

Fig. 5. Feedback control in HRT-HOOD.

lack of an effective analytical framework further compounds the issues. The level of complexity arises from the fact that real-time designs need to be analyzed for schedulability and temporal correctness. The classical object paradigm does not support such analysis on objects nor are objects described by run-time constraints.

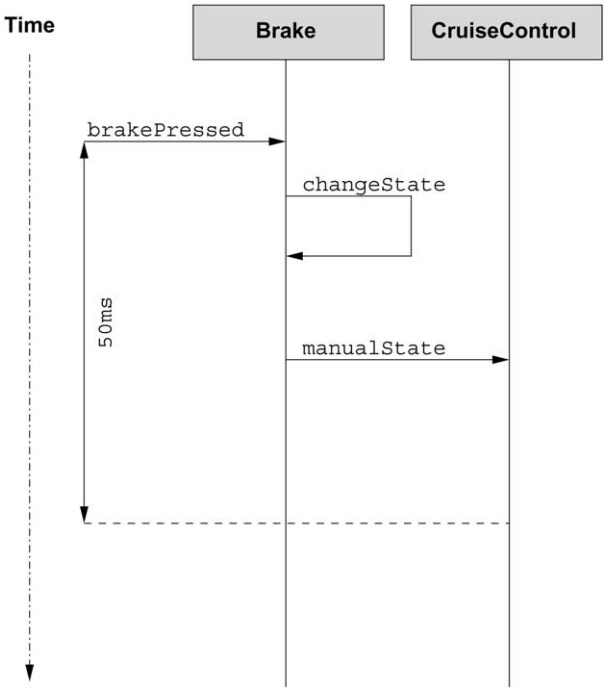

Fig. 6. Brake pressed transaction in HRT-HOOD.

This study illustrates some important attributes and behaviors that real-time objects need to consider so that tasks can be proved schedulable within their time constraints.

1. Real-time objects need to be able to be assigned or collected into tasks or threads.
2. Objects should address timing constraints on their operations by providing attributes for execution time and worst case time for method executions.
3. The operating system of the model should be able to manage the run time priority of the objects.
4. In case of a model, where messages are the mode of communication between objects, a message priority structure should be supported by the model. Note that many operating system kernel support thread priorities and not message priorities. Therefore a kernel that supports message priorities need to be used.
5. When objects communicate via shared data structures mutual exclusion and blocking mechanisms should be enforced by the model.
6. When priority inversions are a problem the model should be able to support the concept of dynamic priorities on object methods.

Finally a suitable tool should be available for creating and analyzing the run-time performance of an application. The model should contain support for performing control, timing and memory management services. Also the model should contain support for managing the dynamic attributes of objects as these attributes continuously change during run-time and are used for meeting response time requirements.

In the ROOM model such services are provided by a virtual machine that is situated between the application layer and the target environment [10].

In this paper, we presented an example case study for evaluating an appropriate real-time object-oriented model from among two possible models. Our study explores some of the issues that need to be addressed when evaluating a real-time model. The main contribution from this paper is the illustration of guidelines for selection of a suitable object-oriented model to represent the system under study. These guidelines can greatly reduce the development cycles for designers by speeding up the model selection process. The issues that we have discussed would help benchmark the various models and tools. Although we did not favor a particular language for representation of these models any modeling language that supports real-time properties and analysis could be used as demonstrated in Sections 3 and 4. This paper reviewed and reinforced the observations made by Saksena and Karvelas [14]. Saksena et al. [5] concluded that the application of real-time analysis techniques to object-oriented design models was not straightforward. They demonstrated that the success of a model depended upon the careful consideration of representing threads, their priorities, scheduling of events within a thread, representation of active objects and message passing mechanisms and the implementation model and its execution environment.

Acknowledgements

We thank Anitha Rajesh for her initial contribution to this project.

References

[1] M. Saksena, Real-time system design: a temporal perspective, Technical Report, Department of Computer Science, Concordia University, 1998.
[2] C. Ekelin, J. Jonsson, Real-time system constraints: where do they come from and where do they go? Technical Report, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 1996.
[3] J. Kratz, Unified Modeling Language for Real-Time Systems Development, Department of Computing Science, University of Nijmegen Toernooiveld 1, Nijmegen, The Netherlands NL-6525 ED, 1999.
[4] T. Bihari, P. Gopinath, Object-oriented real-time systems: concepts and examples, IEEE Computer 25 (12) (1992) 25–32.
[5] M. Saksena, P. Karvelas, Y. Wang, Automatic synthesis of mutitasking implementations from real-time object-oriented models, Proceedings of IEEE Real-time Systems Symposium, 2000.
[6] K. Kim, Object structures for real-time systems and simulators, IEEE Computer 30 (8) (1997) 62–70.
[7] M. Peleg, D. Dori, Extending the object-process methodology to handle real-time systems, Journal of Object Oriented Programming 11 (8) (1999) 53–58.
[8] A. Burns, A. Wellings, Real-Time Systems and Programming: Ada 95, Real-Time Java and Real-Time POSIX, Third ed., Addison-Wesley, Reading, 2001.
[9] A. Burns, A. Wellings, A Structured Design Method For Hard Real-Time Ada Systems, Elsevier Science, Amsterdam, 1995.
[10] B. Selic, G. Gullekson, P. Ward, Real-Time Object Oriented Modeling, Wiley, London, 1994.
[11] R. Grosu, M. Broy, B. Selic, G. Stefanescu. Towards a Calculus for UML-RT Specifications, in: H. Kilov, B. Rumpe, I. Simmonds (Ed.), Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications, Vancouver, Canada, 1998.
[12] G. Booch, Object-oriented development, IEEE Transactions on Software Engineering 8 (2) (1986) 211–221.
[13] L. Cingiser, L. Ma, A UML package for specifying real-time objects, Technical Report, Department of Computing Science, The University of Rhode Island, Kingston, RI, 02881, 1996.
[14] M. Saksena, P. Karvelas, Designing for schedulability: integrating schedulability analysis with object-oriented design, Proceedings Euromicro Conference on Real-Time Systems, 2000.
[15] Y. Ishikawa, H. Tokuda, An object-oriented real-time programming language, IEEE Computer 25 (10) (1992) 66–73.
[16] A. Liu, Real Time Systems, Prentice Hall, New Jersey, 1998.
[17] M. Saksena, P. Freedman, P. Rodriguiez, Guidelines for automated implementation of executable object oriented models for real-time embedded control systems, Proceedings of IEEE Real-Time Systems Symposium, 1997.
[18] A. Burns, A. Wellings, HRT-HOOD: a design method for hard real-time, Real-Time Systems 6 (1) (1994) 73–114.
[19] D. Gaudreau, P. Freedman, Temporal analysis and object-oriented real-time software development: a case study with room/objectime, Proceedings IEEE Real-Time Technology and Applications Symposium, 1996.