A Java-Based Distributed System Framework for Real-Time Development

Jih-Ming Fu, Win-Bin See, Pao-Ann Hsiung[*], *Jen-Ming Chao,* and *Sao-Jie Chen*

Department of Electrical Engineering, National Taiwan University, Taipei, TAIWAN, R.O.C.

[*]Institute of Information Science, Academia Sinica, Taipei, TAIWAN, R.O.C.

Abstract

In recent years, people are trying to make consumer electronics more powerful and have started to embed chips in these products to increase intelligence. Therefore, there should be a powerful application program to control the consumer electronics. For the above reasons, a distributed real-time framework and development environment is proposed, which can be used to produce distributed real-time applications for easily controlling electronics distributed around the world. Our framework consists of three modules: Standalone System, Control Client, and Host Agent. Each standalone system can be controlled by a control client and the host agents act as bridges connecting the standalone systems. Compared to conventional object-oriented application frameworks, our environment is not only modeled in UML, but also network-based, completely written in Java and hence highly portable, remotely controllable, and able to produce customized graphical user-interfaces for applications. Applications developed using the environment show its feasibility as a useful development aid.

Keywords: real-time system, distributed real-time framework, Java, object-oriented application framework, network remote control

1. Introduction

Complex systems, such as aircraft systems, require several powerful application programs to control them. The reason is that every calculation in an aircraft should be in time. Thus, we must design applications having a real-time kernel that can handle multi-task switching in time. This kind of application is called a *real-time system.*

In the 1980's, we did not have many electronic products. But now, every family has at least one electronic facility, namely a telephone set. In the process of improving human material life, lots of consumer electronic products have been and are being made for convenience. For example, every family now has at least a television set, a refrigerator, a washing-machine, and may be a car. In recent years, people are trying to make consumer electronics more powerful and have started to embed chips for increasing intelligence in these products. This results in the need for a powerful application program to control these consumer electronics. This application is usually an *embedded system*, and most embedded systems are real-time systems.

Object-oriented technology has been used in the development of real-time systems for quite some time. Research literature has shown that the concept of objects in real-time systems is useful. Object-oriented real-time (OORT) system models such MO2 [1], evaluation taxonomy such as in [2], object-oriented real-time language design [3, 4], concurrency exploitation in OORT systems using metrics-driven approach [5], checking time constraints [6], and verification of function and performance for OORT systems [7] are some of the recent work on applying object-oriented technology to real-time system design. Although object-oriented technology has been applied to the design of real-time systems in several proposed work, but there has been little work on the development of an object-oriented application framework for real-time system application design, except for *Object-Oriented Real-Time System Framework* (OORTSF) [8], *Real-Time Framework* (RTFrame) [9], and SESAG [10].

Similar to OOAFs' design principles, we develop our distributed real-time framework using UML models and the Java programming language. Our framework has three parts: *Standalone System* (SS), *Client Control* (CC), and *Host Agent* (HA). Each of the three parts contain several layers: from user to physical network. Previous works mentioned above have only described how their OOAFs work and their overall architecture. In contrast, we give details of implementation in Java, which is very useful for system engineers. Our framework mainly differs from previous work in that we propose a method for controlling and connecting distributed standalone systems that otherwise are independent. Our framework has other features also not found in previous OOAFs, such as network connectivity, remote control, graphical user interface customization, and high portability due to the use of Java.

The rest of this paper is organized as follows. Section 2 introduces some background for the whole implementation, such as concepts of real-time system and distributed environment. Section 3 describes the architecture of our system, where we use UML to model our system structure. Section 4 details the system implementation. Section 5 presents an example on our framework environment. We show the ease of using our framework and the emulation environment to communicate between any two standalone systems. Section 6 gives a final conclusion and discusses some future work.

2. Background

2.1 Real-Time System

Typically, a real-time system consists of a controlling system and a controlled system. For example, in an automated factory the controlled system is the factory floor with its robots, assembling stations, and the assembled parts. The controlling system is the computer and human interfaces that manage and coordinate the activities on the factory floor. Thus, the controlled system can be viewed as the environment with which the computer interacts. The activities of controlling systems may cause a disaster if they are not consistent with the actual state of the environment. Hence, periodic monitoring of the environment is necessary. Because of the impact of the controlling systems' activities, the timing correctness requirement is important.

Real-time systems span many application areas. In addition to automated factories, applications can be found in avionics, undersea exploration, process control, robot and vision systems, as well as military application such as command and control. The complexity of real-time systems also spans the gamut from very simple control of laboratory experiments for process control applications, to very complicated projects such as the space station. In these systems, we call the activity a real-time task.

A real-time system is generally specified as a collection of tasks. The tasks are usually independent and periodic. Execution time, period, deadline, type of priority, and resource requirements are specified for each task. Hard real-time systems do not allow the violation of any timing constraint, that is, no task can violate its deadline. Soft real-time system strive to minimize deadline violations. To statically guarantee satisfaction of all timing constraints, the tasks must be scheduled using priority-based scheduling algorithms such as rate-monotonic (RM) [11], earliest-deadline first (EDF) [12], mixed-priority (MP) [12], pin-wheel, etc. or using time-based scheduling algorithms. Dynamic monitoring of timing constraints in distributed real-time systems can be achieved by taking into account the drift among processor clocks [13]. We will be implementing Deadline-Monotonic Scheduling Algorithm (DMSA) [14], Sporadic Server [15], and an algorithm developed by the authors in [16].

2.2 Distributed Environment

With the advance of electronics, a uni-processor computer may have its upper limit in performance. Thus, people have started to design architectures for multiprocessor computing. Since the technique of designing multiple processors in one computer is difficult, the distributed system, where nodes in a network can communicate with each other, becomes more important in recent years. But in a distributed real-time system, communications have to be re-designed for providing a better service constrained by deadlines. To realize this kind of real-time communication, we need high-speed networks, an integration of low-level protocols with the operation system kernel, I/O modules, and application modules. Researchers are proposing various local area network architectures for efficient communication in distributed real-time systems. There has also been significant work in developing reliable atomic broadcasts with varying semantics and performance characteristics. For example, some broadcast protocols support FIFO semantics, others a causal ordering, and yet others are tailored to determine group membership.

Besides, clock synchronization is also a problem in a distributed real-time system. Each processor in a node may maintain its own clock. Even if these clocks are initialized with the same time, since physical clocks drift due to the change in physical conditions like temperature, sooner or later, individual clocks will indicate different times. There are two types of clock synchronization problems. One is *external* synchronization, wherein each clock in a system must be synchronized with the real-world clock. The other problem is *internal* synchronization, which relates synchronization among the multiple clocks within a system so as to keep the relative deviation between individual clock values small.

3. System Architecture

When programmers start to design an embedded system, they should consider everything from the hardware level to the application level. Besides, they may use an assembly language to design the whole system, which is time consuming and hard to understand for non-engineers and even engineers. Thus, we want to use the Object-Oriented (OO) method to model and implement our system.

Our system consists of three modules: (1) *Standalone System*, (2) *Control Client*, and (3) *Host Agent*. In this section, we present the architecture of each module. But first, the whole system architecture is shown in Figure 1. Since these three modules are connected by a network, each of the modules must have a common Network Layer.

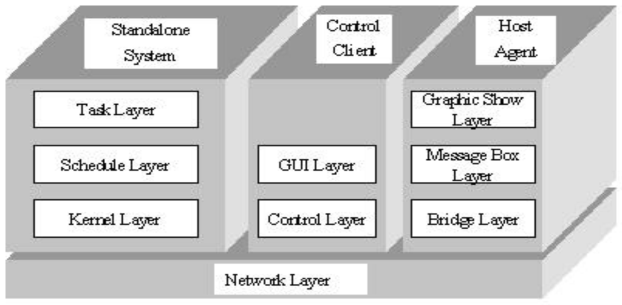

Figure 1. System architecture of the distributed real-time framework

3.1 Standalone System (SS)

The SS is a multitasking real-time system, which can be used to handle sensor, output device, and network communication. A controller can be downloaded from our distributed real-time framework by using a simple web browser and then customized for a specific SS. Therefore, programmers need not care about the GUI (Graphical User Interface) design and the real-time kernel. They just need to think about the design of domain-specific tasks. This module is divided into four layers as discussed in the following:

(1) Task layer

In this layer, four kinds of tasks are defined in our system. First is a task which has a periodic deadline for execution. It is called a *periodic task*. Second is a task which has no periodic deadline, but is executed after some event occurs. It is called an *aperiodic task*. Third is called an *interrupt task* which is executed when some event happens from time to time. Like in a dangerous situation, we must have the ability to close a malfunctioning machine in time. Fourth and the last one is a *native code task*. As our system is written in Java, there might be some special situations that require some functions to be written in some native code, like C or assembly language. We use this kind of tasks to handle the case.

(2) Schedule layer

There is at least one scheduling algorithm on every real-time system. Some of them may have two or more algorithms. In this layer, we can enforce any scheduling algorithm including the one written by a user on the system. Every task will register its information in this layer. All these information will be used to schedule tasks. We can change the scheduling algorithm at any time for damage or some other reasons. But after changing a scheduling algorithm, we must re-schedule all tasks again.

(3) Kernel layer

This layer is the major layer of an SS module. It handles clock ticks, shared memory, and context switches. Because our system is real-time, clocks must be correct at all times. With the correct clock tick, we can handle context switches in time. Shared memory is used by all the tasks created on this system. To prevent incorrect data modification, the multitasking kernel handles mutual exclusion among concurrent tasks.

(4) Network layer

As mentioned, all of the three modules of our system have their own network layers. But, the network layer on each module does not have the same functionality. The network layer in SS module has two major tasks to be done. One is handling incoming commands, and the other one is handling incoming and outgoing data.

3.2 Control Client (CC)

A CC module is used to control a distributed SS module. We use one CC to control one SS. Thus, it should have a user friendly GUI and a control layer between the GUI layer and the network layer.

(1) GUI layer

In order to design a self-created customized GUI, we have to construct some necessary information, like the kind of components, the value of variables, and the range of the variables, to pass to this layer. Using these information, we will construct the correct GUI by building relations between commands and GUI actions.

(2) Control layer

When an action is started, this layer will transform the action into commands and then pass the commands to the network layer for further distribution.

(3) Network layer

This layer is used to receive the information needed by the GUI layer and to send command to the SS module.

3.3 Host Agent (HA)

Host Agent (HA) is the most important module in our system. This HA module serves as a bridge between different standalone systems and as a central controller for all the individual standalone systems. Similar to the home theater system, each SS can send a registry to the HA. In this way, the controller GUI of each SS can be displayed on the HA. With the HA, we can control any consumer product or design a suitable schedule to automatically manage activities in our own home or office. In this module, we should have a layer to handle all the message passing and a layer to control the message passing layer and other layers in the entire environment.

(1) Graphic Show layer

This layer is used to show a comprehensive view of the entire distributed real-time system. When a registration information arrives, this layer will get the information and pass them to the programmer for handling.

(2) Message Box layer

A message box is a queue related to an SS module when this SS starts to register. If there are data transferred to a message box, the box will wake up and transfer the data to the related SS as soon as possible.

(3) Bridge layer

Every transferred data should be received by this layer. Data to be pushed into the correct message box or to be sent to the graphic show layer are controlled by this layer.

(4) Network layer

This layer handles data receiving and pushes them into a receiving queue. Another function is to send a command to a specific SS module.

4. System Implementation

In this section, we start to present the implementation of our system. As described in Section 3, we divide our presentation into three parts. One is the standalone system (SS), one is the control client (CC), and the third one is the host agent (HA). All of these three parts are implemented with JDK version 1.2.

4.1 Standalone System (SS)

SS is an embedded real-time system. As shown in Figure 1, there are four layers in this module.

(1) Task Layer

In our system, we treat every task as a *thread* so as to reduce the overhead of context switching. Thread switching changes only its own stack and register. This is one of the reasons that we implement our system with Java.

Our framework can be used by programmers to create tasks easily. In the system, there is a class, named *TaskJob*, which is an abstract class and has one abstract function, *TaskFun*. By extending the class and implementing the function, a task can be created. Specifically, we derive this class into another concrete class, which can be extended by programmers to handle all these things. Before implementing a class, the following task information must be provided: period time, execution time, deadline, task states, task type, and a unique ID and so on.

All of the above information are used in all the layers of our system. To manipulate these information, we have some useful functions to set all the information of a task.

Having all the above descriptions and information, now we can design a class, *TaskNode*, to include all the above information in this layer and to reduce the work of creating tasks.

In our system, there are four kinds of tasks: *periodic task*, *aperiodic task*, *interrupt task*, and *native code task*. One may assign the type when creating a task. The state diagram presenting task creation and task function running situation is shown in Figure 2.

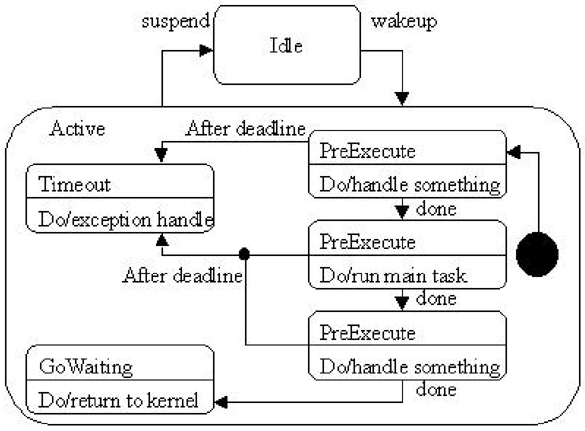

Figure 2. State diagram of task running

Since our system is designed in Java, for programmers who must take advantage of platform-specific functionalities beyond the Java Virtual Machine, if our system can only use the pure Java language, it may not be flexible. To solve this problem, we implement an interface for *TaskNode*. Using this Java Native Interface every method written in a native language (such as C/C++ or assembly) can be loaded by a Java program. This kind of native code task also has three types of conditions inherited from the *TaskNode* class with the *NativeInterface*.

(2) Schedule Layer

A real-time scheduling algorithm is implemented in this layer. We have declared a base class, *Schedule* that is a cyclic scheduling class. With this base class, tasks can be scheduled cyclically. The order of the sequence refers to the task's ID. The task with the smallest ID will be executed first.

In *Schedule* class have a *ReSchedule* method. If *ReSchedule* method perform with time slice in a proper period, then the scheduling algorithm is static algorithm. The dynamic scheduling can perform the *ReSchedule* at any time when it is necessary to change the time slice. The difference of state transitions between the static and the dynamic scheduling as shown in Figure 3.

In our environment, we also extend this base class to implement three scheduling algorithms. One is the *Deadline-Monotonic Scheduling Algorithm* (DMSA) [14], another one is the *Sporadic Server* [15], and the third one is an algorithm designed by us [16]. DMSA is used for scheduling periodic tasks and Sporadic Server is used for scheduling aperiodic tasks. The third algorithm is designed to handle both periodic tasks and aperiodic tasks, but this algorithm is not described here due to page-limit.

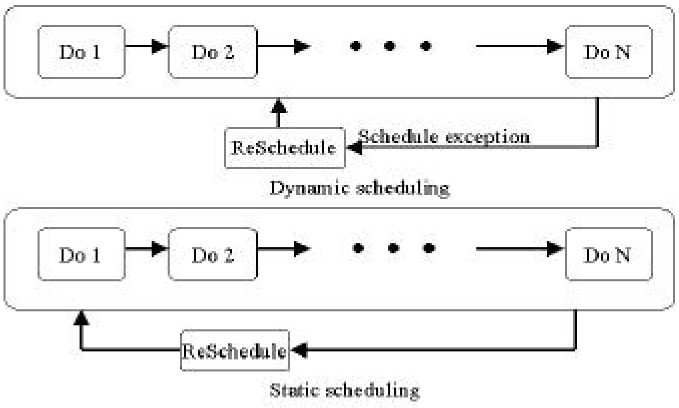

Figure 3. Difference between static scheduling and dynamic scheduling

(3) Kernel Layer

The main components of this kernel layer are clock ticks, task switching, and shared memory. Using clock ticks, we implement the kernel as a class, *Kernel*, which has a *Runnable* interface class. The attributes and behavior of this class are defined as follows:

(4) Network Layer

This layer is controlled by a *Communicate* object with *Register* and *RemoteControl* interface. If the programmers want to implement a distributed real-time system, they must use the *Register* interface to provide information to the HA. *RemoteControl* interface is just a remote method invocation (RMI) object for our system to use.

4.2 Control Client (CC)

This module serves as the remote controller of electronic products. There are three layers in this module. Following are the details of implementation.

(1) GUI Layer

The design of a graphic user interface is not hard but tedious and takes a long time in implementation. To reduce the programmers' job of designing a real-time system and to get the implementation result, we should support a GUI layer in the control client module. To achieve this goal, there must be a *GUInfo* information passing between SS and CC.

(2) Control Layer

This layer is used to implement the relation between a GUI layer and an SS module. The GUI layer must send a correct command to SS through the network layer, and SS must send back all the *GUInfo* information to this layer. This CC layer will record a map of *taskID* and *commandID* and then will call the GUI layer to create a GUI according to the information received.

When we use a remote procedure call (RPC) to execute a command, the CC module will be in the waiting state. After the RPC returns, the CC module will continue to run.

(3) Network Layer

This layer relates to the network layer of the SS. Because we use the remote method invocation (RMI) in Java to implement our network communication, we can just use the class, *RemoteControl* (extended from the *Remote* class), to call the method that we want.

4.3 Host Agent (HA)

A distributed environment can be constructed using this HA module, where an agent is dedicated to handle graphic user interface, message passing, and central management. Following are the details of implementation.

(1) Graphic Show Layer

The goal of this layer is to design a comprehensive interface for the end-users to view in a distributed environment. A programmer has only to extend the *graphicShow* class and use methods in the *graphicShow* to get the correct information from SS.

Since the object created from this class must be executable, it is extended from a *Thread* class. In implementation, a programmer must assign a period to this object and implement the method of *handleMsg*. Then, *handleMsg* will be activated during this period.

(2) Message Box Layer

This layer plays an important role in the HA module. An end-user can control message mapping by using user interface, then the message mapping information will be transferred to the bridge layer. Every message, which is dispatched from the bridge layer, will be directly passed to the related SS.

(3) Bridge Layer

Getting and dispatching messages are the main tasks of this layer. We use a runnable thread to implement it. This thread looks like a watchdog. When there is an incoming message, it will get it from the input queue and dispatch to the related queues according to the *mapTable* in the message box layer. We have also implemented this bridge layer as a class, *Bridge*.

(4) Network Layer

The network layer in this HA module is different from the one in SS or CC. We are using RMI to implement our network communication. The RMI is a kind of remote procedure call (RPC). This network layer serves both as a caller and a callee. This is because an SS registers information by using RPC, and an HA can also use RPC to control the SS. We have implemented this *Bridge* class with two interfaces: *Communicate* and *AgentInterface*.

4.4 System Integration

A computer network is the media for our system integration. The most important thing of a network communication is the interactions among the CC, SS, and HA modules. In this section, we will present all the interactions between these three modules by using state diagram and sequence diagram.

We assume that there are two individual Standalone Systems (SS). Before the environment begins to run, the programmer must know how these two individual systems work. Then, the mapping table is programmed according to the work flow of these two systems.

The control and exchange messages between different real-time systems are passed concurrently. Usually, these two kinds of actions run independently. But they may be activated at the same instant by an SS. Also, individual real-time systems may send messages at the same time.

5. Examples of Application Development

In this section, we will show how to use our framework to design an automatic home control system. We suppose that all the electronic products in a family form a real-time system and are connected by the LAN in a house. With the network and the supported individual processors, we can port our framework to this environment easily.

In this example, assume that we have a phone, a cooler, and an oven. We want to use the phone to activate the cooler and the oven. The phone, cooler, and oven in our system are standalone real-time systems. We still need a host agent to complete our distributed environment. We have defined all the tasks of the systems in Table 1.

Table 1 Definition of the whole environment

System	Task name	Task features
Phone	Listener	(1) hit the number key and activate connection (2) ring the bell if there is a connection (3) automatically answer the phone call
	AutoAnswer	(1) receive the number from the remote phone (2) change the numbers to mapping command (3) send a command to host agent and then bypass to other systems
cooler	SetTemp	Set the temperature of the cooler
	SetStartTime	Set the start time of the cooler
	SetStopTime	Set the stop time of the cooler
	Receiver	Receive the command from the phone
oven	SetBurn	Set the value of burn on the oven
	SetTimer	Set the timer for the oven
	Receiver	Receive the command from the phone

6. Conclusion

We have designed a package which can help real-time system designers to develop their real-time applications easily. There are three real-time scheduling algorithms for a programmer to use in this package: Deadline-Monotonic Scheduling Algorithm, Sporadic Server, and another algorithm designed by ourselves. We use threads to implement the lightweight tasks in a real-time system. In this way, we can reduce the overhead of task switching. Besides, this package also includes a remote controller (control client module) that the programmer can quickly use with minimal efforts. This kind of remote controller can be downloaded with a browser. So, the user can control the system anywhere. We have also integrated and implemented the package to support a distributed environment, where we use a host agent as a bridge to transfer messages between individual real-time systems. This host agent can limit the network transfer time during communicating between individual systems if all the systems and the host agent are in the same LAN. Our environment is designed in 100% pure JAVA and we use UML to model it. This makes our environment more portable, more flexible, and more understandable.

References

[1] A. Attoui and M. Schneider, "An object oriented model for parallel and reactive systems," in Proc. Real-Time Systems Symposium, pp. 84-93, Dec. 1991.

[2] D. Hammer, L. Welch, and O. van Roosmalen, "A taxonomy for distributed object-oriented real-time systems," ACM OOPS Messenger, Special ISSUE on Object-Oriented Real-Time Systems, Vol. 7, pp. 78-85, Jan. 1996.

[3] Y. Ishikawa, H. Tokuda, and C. W. Mercer, "Object-oriented real-time language design: constructs for timing constraints", ACM SIGPLAN Notices, ECOOP/OOPSLA' 90 Proceedings, Vol. 25, pp.289-298, Oct. 1990.

[4] B. Achauer, "Objects in real-time systems: Issues for language implementers," ACM OOPS Messenger, Vol. 7, pp. 21-27, Jan. 1996.

[5] L. R. Welch, "A metrics-driven approach for utilizing concurrency in object-oriented real-time systems," ACM OOPS Messenger, Vol. 7, pp. 70-77, Jan. 1996.

[6] M. Gergeleit, J. Kaiser, and H. Streich, "Checking timing constraints in distributed object-oriented programs," ACM OOPS Messenger, Special Issue on Object-Oriented Real-Time Systems, Vol. 7, pp. 51-58, Jan. 1996.

[7] J. Browne, "Object-oriented development of real-time systems: Verification of functionality and performance," ACM OOPS Messenger, Special Issue on Object-Oriented Real-Time Systems, Vo l. 7, pp. 59-62, Jan. 1996.

[8] W.-B. See and S.-J. Chen, "High-level reuse in the design of an object-oriented real-time system framework," in Proc. International Computer Symposium, pp. 363-370, Dec. 1996.

[9] P.-A. Hsiung, "RTFrame: An object-oriented application framework for real-time applications," in Proc. 27th International Conference On Technology of Object-Oriented Languages and Systems, p. 138-147, IEEE Computer Society Press, Sep. 1998.

[10] P.-A. Hsiung, "Object-oriented application framework design for real-time systems," in Proc. 4th International Symposium on Real-Time and Media Systems (RAMS'98), pp. 221-227, Taipei, Taiwan, Sep. 1998.

[11] J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in Proc. Real-Time Systems Symposium, pp. 166-171, 1989.

[12] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment," Journal of ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[13] F. Jahanian, R. Rajkumar, and S. C. Raju, "Runtime monitoring of timing constraints in distributed real-time systems," Journal of Real-Time Systems, Vol. 7, No. 3, pp. 247-271, 1994.

[14] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," Performance Evaluation, Vol. 2, No 1, pp. 237-250, 1982.

[15] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard real-time systems," Journal of Real-Time Systems, Vol. 1, No. 1, pp. 27-60, 1989.

[16] Jen-Ming Chao, "Design and implementation of a support environment for distributed real-time system," Master Thesis, Graduate Institute of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC, Jun. 1999.