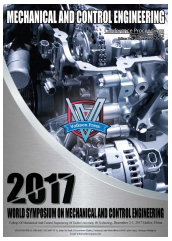
ISBN: 978-1-948012-06-5



Contents List available at VOLKSON PRESS

World Symposium on Mechanical and Control Engineering (WSMCE)

RESEARCH AND APPLICATION OF RTSJ-BASED EMBEDDED SYSTEM API

Teng Haikun*, Liu Xinsheng, Wang Shiying

Computer and Information Engineering College, Heihe University Academic Road No. 1, Heihe City, China
****Corresponding Author Email: thk_1983@163.com***

ARTICLE DETAILS

Article History:

Received 02 october 2017
Accepted 06 october 2017
Available online 11 november 2017

Keywords:

RTSJ, API, concurrent real-time calculation structure, Java Processor

ABSTRACT

This paper describes a RTSJ-based application program interface which aims to provide advanced concurrent real-time calculation structure, facilitating real-time embedded system development. Further, development and optimization (the time/footprint requirement) of real-time Java applications to a specific Java processor are carried out. The use of the proposed API is illustrated in the paper by means of a case study that implements a crane control system. This case study highlights the benefits and advantages of the proposed API.

1. Introduction

With the rapid development and the wide application of computer technology, network communication technology and the micro-electronics technology, the real-time embedded systems (RTES) has deep into every field of scientific research and social life. The conventional wisdom is that real-time embedded systems generally use outdated or obscure languages such as C/C + +, assembly language, etc. These development means complicated programming and disadvantages such as low efficiency, easy to get wrong, and lack of security has become the development of embedded system, and a new generation of real-time embedded systems look forward to more new features, which greatly increased the complexity of the embedded system, so we need new development language and means to promote the development and application of real-time embedded systems. Fortunately, the Java technology to make up for the characteristics of the above shortcomings, and become more and more get the favour of embedded programming, Java real-time specification (RTSJ, also known as the Java specification application (JSR - 001) is the best example, real-time Java provides real-time programmers with a designed for productivity, the mainstream of modern language [1-2].

Java real-time specification is real-time Java (RTSJ) panel (RTJEG) of Java extension specification in real time, make up for the defects of the Java language in real-time applications, it provides the creation, validation, analysis, implementation and management of real-time Java thread application program interface (API). At present, there are many Java platforms that support RTSJ and mature commercial products. For example, TimeSys launched the first industrial version of the embedded Java platform JTime, which conforms to the RTSJ, and the jRate runtime system based on the RTSJ extension of GCJ (GNU Compiler for Java) [3,4]. This system with other real-time Java platform is somewhat different, because jRate Java application source code to compile (AOT) generates native code, which means that don't need a Java virtual machine, saved a lot of unnecessary spending. However, these implementations are not aimed at the real-time embedded domain, because the overhead footprint (footprint) requirements in embedded systems are just as important as real-time requirements. The aim of this article is using configurable embedded Java processor FemtoJava executes Java bytecode, and optimally effective operation code necessary for the execution of the application, custom Java applications can by Sashimi development environment to compile with VHDL form processor core, Sashimi can be automatically converted FemtoJava processor architecture [5]. FemtoJava is according to the specific application and design based on FPGA technology reduced instruction of harvard structure place stack principle, it contains more than one multiplexer and register, and a unique ALU, more suitable for frequent changes in application fields.

However, the Sashimi environment lacks a programming model that represents concurrent and real-time constraints. The main goal of this article is to make up for the lack of the Sashimi environment by providing an API based on RTSJ to support concurrent task specifications, which support concurrent task specifications and time limit specifications. In order to overcome the limitation of Sashimi environment, this paper has made some modifications to the RTSJ specification. On the Sashimi environment using the provided API, programmers can develop concurrent real-time applications, and deploy them to FemtoJava processor, implementation is based on the RTSJ API in the practical application of embedded system.

2. THE SASHIMI ENVIRONMENT

Sashimi development environment is a free and effective JVM optimization tool for embedded systems that developers can use to Java simulation, emulation and directly implement embedded systems [5]. The Sashimi environment can support concurrent tasks by extending the API, implementing the RTSJ standard. According to the Sashimi environment; designers can use the Java language to develop their applications directly. In order to meet the constraints of Sashimi environment, some programming restrictions must be followed. For example, programmers can only use apis provided by Sashimi environment rather than standard Java application interfaces. In addition, designers can only use static methods and properties, because the Sashimi environment object does not support dynamic allocation, and class hierarchy method of inheritance and polymorphism, and the basic concept of object-oriented development support. In the Sashimi environment, the Java source code uses the standard Java compiler to generate Java bytecode. These generated classes can be tested on the host platform using the API class library that emulates the Sashimi environment. The next work, based on the generated Java bytecode, comprehensive application and FemtoJava processor generates a customized FemtoJava CPU control unit, the control unit is only supported opcodes used by applications. The size of the control unit is proportional to the number of different operating codes utilized by the application software, making it suitable for embedded system applications.

In addition, this article needs to extend the API of Sashimi environment to allow concurrent programming and add an operating system layer of

dynamic task scheduling, so that Sashimi environment supports different scheduling algorithms. In terms of expenditure footprint, energy consumption, and real time performance, the impact assessment of these algorithms is described in detail in reference to [6]. The downside of Sashimi's environment is the lack of high-level real-time architecture, leading designers to use low-level system calls to generate concurrent processes and interact with the scheduler. In addition, the Sashimi environment has no mechanism for clearly expressing task time constraints. These issues are handled through the API provided in section.

3. THE PROPOSED API

As mentioned earlier, the main purpose of the development API is to provide advanced real-time programming architecture support for configurable embedded system hardware/software structures based on FemtoJava micro-controllers. These APIs are based on Java real-time specification [1]. It allows you to use a scheduling object, which is an instance of the scheduler interface class, like the Realtime Thread (real-time thread) class. At the same time; it also provides a set of class storage parameters that represent specific resource requirements for one or more scheduling objects. For example, the Ralease Parameters class (the release parameter class, the parent of Aperiodic Parameters and Periodic Parameters) contains some useful parameters to meet the real-time requirements specification. In addition, the RTSJ API also supports the following concepts: time values (absolute time and relative time), timers, periodic/sporadic, non-periodic tasks, and scheduling policies. The term 'task' represents a scheduling member of the system context, or a scheduling object.The following are simple descriptions of the main classes.

Real-time Thread: the real-time thread class extends the default java.lang. thread class. Real-time Thread class in real time embedded system represents a real-time task, which is either periodic or non-periodic depending on the release parameter object given. If the Periodic Parameters type of the task is used to release parameters, the task is periodic; if the task USES an instance of Aperiodic Parameters or the Sporadic Parameters class, the task is sporadic and non-periodic.

Release Parameters: the release parameter class is the base classes for all the release parameters of the real-time task. Instances of the Release Parameters class include the cost of publishing, the startup time, and the processor or cost overruns that miss the deadline. Its subclasses, Periodic Parameters and Aperiodic Parameters, respectively represent the release parameters of periodic and non-periodic tasks. Periodic Parameters must contain a period value and the start/end time value. The Sporadic Parameters class, a type of non-periodic task, inherits the Aperiodic Parameters class, and the running cycle of the occasional task equals the minimum of its two interval intervals.

Scheduling Parameters: the scheduling parameter class is the base class for all scheduling parameters used by the scheduler object. The Priority Parameters class represents the priority of the task, whose instances are assigned to the Schedulable object, whose execution credentials are determined by the priority. Designers simply create a new instance or share an existing instance, provide an integer value to the constructor as a priority, and assign to existing Schedulable.

Scheduler: the Scheduler itself is an abstract class. Its subclass "Priority Scheduler", "Rate Monotonic Scheduler" and "EDF Scheduler" respectively represent fixed priority, Rate Monotonic and the earliest time-priority scheduling algorithm.

High Resolution time: compared with the java. util. date class, the RTSJ provides strong support for the concept of time. The high-resolution time class is an abstract class that cannot be instantiated. However, it stores milliseconds and nanosecond fields for all other high-resolution time classes and provides methods for its subclass Absolute Time (absolute time), relative time (relative time). Absolute time is given by an offset, the reference value is Greenwich time; Relative time is always a period of time. It can take integers, negative Numbers, or zero. In addition, the rational time class extends the Relative Time class by increasing the frequency, which expresses the frequency at which something happens at each interval.

Clock: RTSJ supports multiple Clock concepts, and the Clock API based on RTSJ defines the real-time Clock expression global Clock reference. The clock class returns an absolute time object representing the current date and time of the system.

Timer: the Timer class is an abstract class that represents the system Timer. Its subclass One Shot Timer and Perodic Timer respectively represent one-time timers and periodic timers.

The implementation of some API classes in this article is slightly different from the way the RTSJ recommended, due to the limitations of FemtoJava processor architecture. This difference exists in Realtime Thread classes, where two abstract methods of the real-time thread class must be implemented in its subclasses—main Task () and exception Task (). They represents the task body (the run () method of the normal Java thread) and the exception handling code that misses the deadline. The latter replaced a Async Event Handler (asynchronous event handler) object, in the RTSJ asynchronous event handler is designed to deal with real-time applications may need to deal with different systems and programming defined events, and asynchronous event handler object should be passed to the release of the parameter object. If the task misses the deadline, an exception is thrown and the exception Task () function is executed. After the exception handling code is executed, the task execution process may jump to the run () method or terminate, depending on the characteristics of the real-time task. If the task is periodic, the run () method should be restarted. This difference suggests using the scheduling algorithm for task pair concepts.

In section 1, the traditional version of the Sashimi environment does not support object creation. As a result, we need to extend some of the new attributes to the Sashimi environment in order to provide full support for apis developed on FemtoJava platform. First, you need to extend the integration of Sashimi environment support objects. According to the relevant changes, application objects are allocated statically at the time of composition. In other words, all objects in the system define a priority, and the entire memory is allowed to be detected for the convenience of storing objects in RAM. Although it may involve high memory usage, but it is very suitable in the real-time system development approach, because it avoids the use of the garbage collector (GC), no time limit, do not interrupt the garbage collector algorithm, makes the Java to run the program is quite lack of certainty, the uncertainty in the real-time embedded system should not be tolerated.

FemtoJava microprocessors introduce four new opcodes to support the provided APIs: getfiel, putfield, invokevirtual, and invokspecial. The first two operators are related to the access to the object area, and their functions are to acquire and set values. The other two are related to method invocation, and the invokevirtual operation code is used to invoke public or protected methods; The invokspecial al operation code is used to invoke the constructor and private methods. Another extension of the FemtoJava microprocessor is the addition of a real-time clock to provide time concepts for embedded systems. Both API members and scheduling layers based on the RTSJ can use this clock.

4. APPLICATION OF API IN REAL EMBEDDED SYSTEM

This paper uses the elevator control system as a case to verify the expansion of Sashimi environment, which is a system for optimizing the scheduling of multiple elevators, including the concurrent tasks and hard real-time constraints [7]. Because of the diversity of elevator control system tasks, traditional development means will be difficult to solve the combination optimization problem of online scheduling and resource allocation. This article USES real-time Java multithreading technology to change these situations. Figure 1 shows the elevator control system of the kind of collaboration diagrams, we can observe from the figure, the Timer class implements the Sashimi environment task time constraints, through the get Real time Clock () method call real time clock single object to obtain real time clock reference.

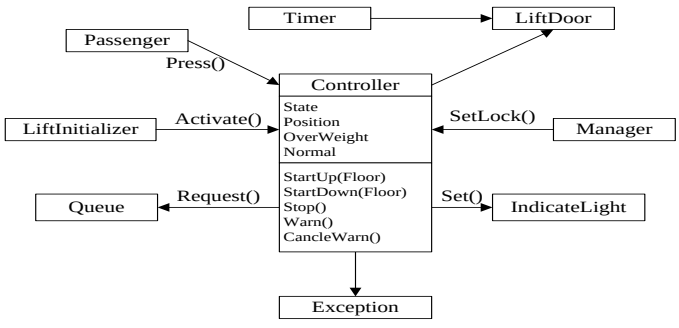


Figure 1: Class collaboration diagram of elevator control system

Although there are many classes in the elevator control system, this article will focus on "lift initializers" and "Lift Door" classes. Because the authors

consider these two classes to be sufficiently representative in the use of apis. Lift Initializer class is the main class of elevator control system. The function of this class is to be responsible for object allocation, initialization, and startup (real-time task application). From the elevator control system code, we can see that only static objects are allocated. The initSystem () method represents the starting point of the application execution process and provides the function of object initialization and real-time task start. Real-time task start can be implemented by calling the start () method. The last call to the sleep () method, which means that the initialization class is no longer used, and is locked. This approach is only used in system initialization methods. The Lift Initializer code of the elevator control system is as follows:

```
public class LiftInitializer        {
/ / application object allocation
public static Controller  nominalCtrl        =              new Controller();
public static Passenger  passenger  =  new Passenger ();
public static LiftDoor  liftdoor   =              new LiftDoor ();
public static Queue  queue        =              new Queue ();
public static IndicateLight  indicatelight  =  new IndicateLight();
public static void initSystem()   {
.../ / object initialization
/ / real-time mission startup:
Lift.nominalCtrl.start();
.../ / start other tasks
While (true) FemtoJava.sleep()；
  }
};
```

With regard to the "Lift Door" class, the associated pattern in the system represents a concurrent real-time task, which is closely related to the system clock. This article uses a custom different application. It is important to emphasize the FemtoJava processor generated by the optimized Sashimi environment, because it only supports Java code for embedded system software. Another thing to note: once the application objects are allocated within the integration time, there is no need to use the garbage collector. While these can lead to higher memory consumption; it provides the deterministic requirements for real-time embedded systems.

5. CONCLUSION

The goal of this article is to optimize real-time embedded system development using the RTSJ API, which is the target platform for FemtoJava processors dedicated to Java bytecode. These API provides programmers with the tools necessary to solve the virtual machine and the application of variability, and programmers can be used in real-time applications said senior mechanism concurrent and timing constraints. In order to ensure that the API is as close to the RTSJ specification as possible, this paper makes minor modifications to the RTSJ specification, which further enhances the quality of the embedded real-time service. Meanwhile, the advantages of real-time Java will greatly change the design difficulty of embedded control software. In the years to come, real-time Java technology will have a huge impact on embedded control.

ACKNOWLEDGEMENT

This paper is supported by Heilongjiang Province Colleges and Universities Basic Scientific Research Service Charge HEIHE University Special Fund Project (ID: 18KYYWFRC04).

REFERENCES

[1] Bollella, G., Gosling, J. 2011. The Real-Time Specification for Java [OL].04. http://www.rtj.org/rtsj-V1.0.pdf.

[2] Eric, J.B., Greg, B. 2010. Java real-time programming [M]. Tian Siyuan (translated). BeingJing: Mechanical industry press.

[3] Haikun, T. 2014. Design of RTSJ-Based Intelligent Home System Gateway [J]. BioTechnology: An Indian Journal, 10 (9), 3937-3943.

[4] Angelo, C., Douglas, C.S. 2012. The Design and Performance of the jRate Real-time Java Implementation[C]// the 4th International Symposium on Distributed Objects and Applications. Irvine, CA, October-November.

[5] Ito, S.A., Carro, L., Jacobi, R.P. 2011. Making Java Work for Microcontroller Applications [J]. IEEE Design and Test of Computers, 18 (5), 100-110.

[6] Becker, L.B., Wehrmeister, M.A., Carro, L., Wagner, F.R., Pereira, C.E. 2014. Evaluating High-level Models for Real-Time Embedded Systems Design[C]// 29th Workshop on Real-Time Programming. Istanbul.

[7] Jianzhong, C., Fei, L. 2012. The modeling and control strategy of a new elevator group control system [J]. Microcomputer information.