Performance analysis of the Sporadic Server Implementation in Real-Time Specification for Java

Carlos M. Tripode, Rodrigo Santos, Javier Orozco
Dep. Ing. Eléctrica y de Computadoras – Universidad Nacional del Sur
Instituto de Investigaciones en Ingeniería Eléctrica – CONICET
Avda. Alem 1253, 8000 Bahía Blanca – Argentina.
Bahía Blanca, Argentina

***Abstract*—In this paper, we introduce the implementation of the Sporadic Server algorithm at the user level in Real Time Java. That is, we do not modify the specification of the Java Virtual Machine (JVM) so our proposal runs without problems on every JVM with support for real-time scheduling. The paper in- cludes an experimental evaluation of the aperiodic handling mechanism using as benchmark the one developed for the RTSJ**

I. Introduction

Real-time systems programming is a very important issue. The need to provide temporal guarantees for time critical systems turns the design and implementation of systems with time constraints many times a custom process. In fact, real-time operating systems are just becoming popular and common in the last years. In the past, real-time systems were developed from the application level all the way down to the hardware level, so every piece of code was under control in the development process. Changing the hardware platform supposed also a change in the software and most of the time a complete revision of the code was necessary. Recently, several real-time operating systems providing specific primitives and scheduling interfaces for the applications were developed and a whole new approach to the development of real-time systems is now possible. Moreover, there are several open-source operating systems and real-time frameworks that provide developers with tools for creating applications in a simpler way.

Hardware independent languages like Java were not used widely for the imple- mentation of control applications because of low predictability, no real-time garbage collection implementation and cumbersome memory management [1]. However, this has changed in the last years with the definition and implementation of the Real-Time Specification for Java. In 2002 the specification for the real-time Java (RTSJ) was approved. The first commercial implementation was issued at the Spring of 2003. In 2005, the RTSJ 1.0.1 was released together with the Real-Time Specification (RI). In September 2009 Sun released the Java Real-Time System 2.2 version which is the latest stable one. Although there have been many papers on embedded systems implementa- tions based on RTSJ and even a full Java microprocessor (implementing the bytecodes) on an FPGA chip has been proposed and used [2], its use as a development language for real-time systems is not generalized.

The introduction of aperiodic or non-real time tasks within a real-time system is becoming every day more important. Basically, we can say that the real-time sys- tem is hide to the common user by the aperiodic applications. For example, modern cell phones or smart phones have many applications. Obviously, it has a real-time core system for handling the digital processing necessary for keeping a live conversation. However, in the background the user may want to keep the e-mail updated, surf the web, play some simple game or read an e-book. All these applications add value to the product and are in many cases the reasons for choosing one or another model. The user sup- poses that the telephone will work as that whatever the model he chooses, but wants additional services. It is in these cases that a proper algorithm to handle mixed applications (real and non real-time) becomes important. Reservation mechanisms have been adopted as the main strategy for handling non real-time aperiodic requests in the form of severs. Among them we can mention the Sporadic (SS) and Deferrable (DS) Servers [3], [4] used with fixed priority (FP) scheduling and the Constant Bandwidth Server [5] used with earliest deadline first (EDF) scheduling.

II. RTS Java Specification Summary

The Real-Time Specification for Java [6], [7] provides a framework for developing real-time scheduling mostly on uniprocessors systems. Although it is prepared to support a variety of schedulers only the PriorityScheduler is currently defined and it is a preemptive fixed priorities one with 28 different priority levels. These priority levels are han- dled under the Schedulable interface which is implemented by two classes: RealtimeThread and AsyncEventHandler (AEH). The first one, is a subclass of java.lang.Thread, so its execution is associated to only one thread. It adds ac- cess to real-time services such as asynchronous transfer of control, non-heap memory (memory handling is explained later), and advanced scheduler services [6]. The second one, encapsulates code that is released after an instance of AsyncEvent to which it is attached occurs. This class creates dynamically a thread.

Traditional Java uses a Garbage Collector (GC) to free the region of memory that is not referenced any more. The abstract class MemoryArea models the mem- ory as divided in regions. The RTSJ defines a subclass NoHeapRealtimeThread of RealtimeThread in which the code inside the method run()

978-0-7695-4641-4/11 $26.00 © 2011 IEEE
DOI 10.1109/SBESC.2011.15

should not reference any object within the HeapMemory area (traditional Java). With this, a real-time thread will preempt the GC if necessary.

III. Sporadic Server Implementation

RTJ provides the class AsyncEventHandler to handle Aperiodic events. A complete description of its functionality can be found in [8]. The mechanism is robust in a certain way but involves the creation of a pool of RealtimeThreads, a priority queue and dynamic binding between the AsyncEvents, the handlers and the threads. Even if this implementation is robust and has been proven correct, there is no control on the aperiodic load, the schedulability conditions and the blocking on other true real-time threads.

The use of a Sporadic Server with a budget, period and priority well defined before hand provides predictability on the way in which the aperiodic events will be handled and this fact is as important as having a short response time. The SporadicServer class is implemented as an extension of RealtimeThread. Thus it is a schedulable object that can be chosen by the scheduler whenever it is ready and has enough priority. AsyncEvents are binded to the SporadicServer in a very simple way. In Figure 1 the way in which the Sporadic Server is defined it is shown.

```
public class SporadicServerRT extends RealtimeThread
{
protected PeriodicParameters periodicParameters;
private TaskQueue aperiodicPending;
private Queue<Replenishment> replenishmentPending;
private int MAX_REPLENISHMENT_PENDING = 10000;
private boolean fairness = true;
protected final Object lock = new Object();
private List listWithAperiodic = new LinkedList();
private RelativeTime actualCapacity;
private RelativeTime consumedCapacity;
.....
```

Figure 1. SporadicServer definition

```
public SporadicServerRT(SchedulingParameters sp,
ReleaseParameters rp,
MemoryArea ma)
```

Figure 2. Sporadic Server class constructor

The replenishments are handled through an AsyncEventHandler. The re- plenishment instants depend on the execution history of the aperiodics and the capacity of the Sporadic Server. In Figure 3 the class Diagram for the Sporadic Server is shown.

IV. Experimental Evaluation

The implementation of the Sporadic Server was evaluated with two different experiments. In the first one, the aperiodics arrivals follow a uniform distribution with minimum inter-arrival time greater than the server period. In fact, the server should be able to handle this kind of aperiodic load easily. In the second experiment, we use a burst distribution for the aperiodics. In this case, the response time of the Sporadic Server will not be as good as in the previous case because

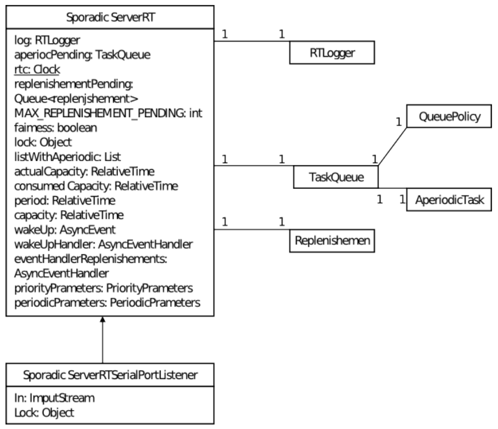


Figure 3. Class diagram

the server will have to recharge its capacity postponing the execution of the aperiodic at least for the period of the server once its budget is depleted.

The evaluation was made on two aspects: latency and response time. The ex- periments were built based on a Sun Real-Time Java Virtual Machine running on top of GNU/Linux Ubuntu 10.04 and Real-time Kernel 2.6.31.11 The system has 5 periodic real-time threads with different computational loads (C) for different utilization factors of the system (0.4, 0.75). The performance of the Sporadic Server was considered for the different real-time loads. Table 1 shows the basic configuration of the system expressed in milliseconds. The worst case execution time of the periodic tasks is chosen from the interval indicated in the column of C to reach the desired utilization factor.

Task	C	T
1	6-12	120
2	12-28	150
3	15-32	200
4	25-48	250
5	30-57	300

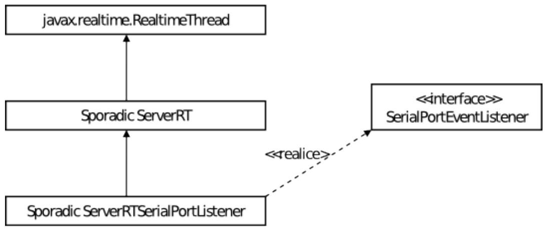
Table I
System Configuration

The aperiodic load was fired in both experiments by the arrival in the serial port of traffic. This traffic was used as an AsyncEvent to fire the execution of the aperiodic tasks. In order to have a common reference for comparison, in both cases the aperiodic execution time was limited to 5 ms which is the minimum sporadic server capacity.

Figure 4 shows the Inheritance Diagram for the Sporadic-ServerRTSerialPortListener that implements the reception of the traffic in the serial port and dispatches the aperiodic task associated.

The evaluation of the Sporadic Server is based taking the ApeiodicEventHandler provided by the Java Specification as a reference.

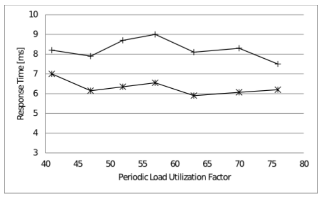
Sparse Arrivals: Figure 5(a) shows the average response time as a function of the peri- odic load in the system. Figure 5(b) shows the latency for each request. As it can be seen



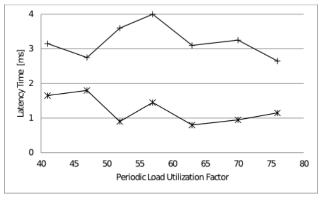
javax.realtime.RealtimeThread
Sporadic ServerRT
<<interface>> SerialPortEventListener
<<realice>>
Sporadic ServerRTSerialPortListener

Figure 4. Inheritance Diagram for the PortListenerClass

the performance of the Sporadic Server is almost constant independently of the periodic load or the capacity of the server.



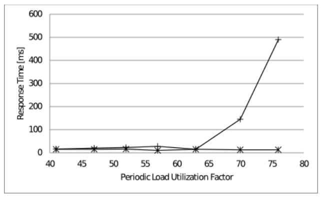
(a) Response time



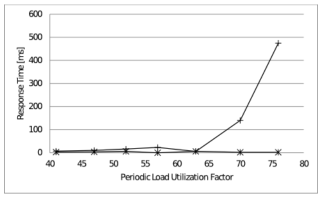
(b) Latency

Figure 5. Sparse Arrivals, + Sporadic Server * AEH

Burst Arrivals: Figure 6(a) depicts how the behavior of the Sporadic Server becomes worst as the capacity of the server is reduced when the periodic load is increased. What happens with the response time also happens with the latency since both parameters are related. Figure 6(b) shows the behavior of the latency in this case. **Results Explained:** The behavior of the Sporadic Server depends on two aspects: its own parameters and the aperiodic demand. When both aspects are well related, that is when the average bandwidth demand of the aperiodics is lower than the utilization factor of the server and also the average execution time of the aperiodics is lower than the capacity of the server, the latency and response times are bounded. Moreover, they can be predicted and are constant in time independently of the periodic load. The burst situation is handled well by the Sporadic Server only in the cases that the capacity of the server is enough to satisfy the demand of the aperiodics in the burst.



(a) Response time



(b) Latency

Figure 6. Sparse Arrivals, + Sporadic Server * AEH

V. Conclusions

In this paper we have presented the Sporadic Server implementation for the Real-Time Specification of Java at the application level. The theoretical aspects of the imple- mentation has been discussed. The Sporadic Server is defined at user level as another RealtimeThread, it has a very simple structure that can be used for dispatching ape- riodic tasks in a dependable way. The use of the Sporadic Server, guarantees that no overruns occur and what is more important that the aperiodics do not make a real-time thread miss its deadline. The solution has been evaluated experimentally.

Even though if there is a complete real-time specification which is, as has been shown in this paper, quite flexible, the use of Java for developing embedded applications is not yet widespread. The introduction of specific Java platforms like cell phones will in the near future probably change this tendency and in the next years more and more real-time Java applications will be developed. In this paper, we have presented a simple implementation. As future work we intend to apply this knowledge in the development of applications for this kind of systems.

References

[1] S. G. Robertz, R. Henriksson, K. Nilsson, A. Blomdell, and I. Tarasov, "Using real-time java for industrial robot control," in *Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems*, ser. JTRES '07. New York, NY, USA: ACM, 2007, pp. 104–110. [Online]. Available: http://doi.acm.org/10.1145/1288940.1288955

[2] M. Schoeberl, *JOP Reference Handbook: Building Embedded Systems with a Java Processor*. CreateSpace, 2009, no. ISBN 978-1438239699, available at http://www.jopdesign.com/doc/handbook.pdf. [Online]. Available: http://www.jopdesign.com/doc/handbook.pdf

[3] B. Sprunt, "Aperiodic task scheduling for real-time systems," Ph.D. dissertation, Department of Electrical and Computer Engineering Carnegie Mellon University, 1990.

[4] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard-real-time environments," *IEEE Transactions on Computers*, vol. 4, no. 1, January 1995.

[5] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the 19th IEEE RTSS*. Madrid, Spain: IEEE Computer Society, 1998.

[6] S. Microsystems, "Real-time specification for java documentation," http://www.rtsj.org/.

[7] J. Gosling and G. Bollella, *The Real-Time Specification for Java*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[8] M. S. Kim, "Asynchronous event handling and the real-time specification for java," Ph.D. dissertation, Dep. Computer Science, University of York, 2009.