J. Parallel Distrib. Comput. 74 (2014) 1930–1944

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Resource management policies for real-time Java remote invocations

Pablo Basanta-Val *, Marisol García-Valls

Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Spain

HIGHLIGHTS

- An integrated technique to manage remote invocations in Java's RMI.
- Empirical evidence on the performance given by the techniques.
- An identification of other alternatives that may benefit from the techniques.

ARTICLE INFO

Article history:
Received 27 September 2012
Received in revised form
25 July 2013
Accepted 1 August 2013
Available online 14 August 2013

Keywords:
Real-time Java
Real-time middleware
RT-RMI
RTSJ
DRTSJ

ABSTRACT

A way to deal with the increasing cost of next generation real-time applications is to extend middleware and high-level general-purpose programming languages, e.g. Java, with real-time support that reduces development, deployment, and maintenance costs. In the particular path towards a distributed real-time Java technology, some important steps have been given into centralized systems to produce real-time Java virtual machines. However, the integration with traditional remote invocation communication paradigms is far from producing an operative solution that may be used to develop final products. In this context, the paper studies how The Real-Time Specification for Java (RTSJ), the leading effort in real-time Java, may be integrated with Java's Remote Method Invocation (RMI) in order to support real-time remote invocations. The article details a specific approach towards the problem of producing a predictable mechanism for the remote invocation – the core communication mechanism of RMI – via having control on the policies used in the remote invocation. Results obtained in a software prototype help understand how the key entities defined to control the performance of the remote invocation influence in the end-to-end response time of a distributed real-time Java application.

1. Introduction

Analyzing the evolution of real-time technology – looking at [58] to study its past evolution and at [34,46,64] to identify the future – we appreciate how real-time systems seem to follow a path parallel to the one previously covered by general purpose applications. If in the very early stages of its existence, influenced perhaps by the high cost of hardware, the search for optimal and efficient algorithms monopolized most research efforts; nowadays, other issues more related to portability, adaptability, and maintainability are gaining momentum. Indeed, whereas primitive real-time systems were embedded and isolated pieces of code, the current generation of real-time systems is more open and COTS (commercial-off-the-shelf) real-time operating systems and middleware are commonly used [46,52].

In this continuous reinvention, some eyes are fixed on the use of high abstraction programming languages – such as Java [48] – as a new way to reduce development costs not only in general purpose applications but also in real-time systems [11,48]. The use of Java attracts attention on specific topics such as byte-code efficiency, real-time portability, predictable code downloading, and especially on automatic memory management. The financial benefits may be quite interesting too—some reports on general purpose productivity [50,51] promise improvements ranging from 20% to 140% when instead of C++ or ADA, developers choose Java.

In the specific area of real-time Java, extraordinary efforts have been carried out for centralized systems with the definition of The Real-time Specification for Java (RTSJ) [31]. This specification is backed by a number of commercial implementations: IBM [1], Oracle [2], and Aicas [3] ready for developing applications. However, one of the main efforts in the area of distributed systems: The Distributed Real-Time Specification for Java (DRTSJ) [43] – based on the RTSJ and Java's Remote Method Invocation (RMI) [4] – is still far from producing commercial implementations [16,41]. In essence, the DRTSJ sets the focus of its activity on the creation of a framework for distributable real-time threads [13,35,63]: entities that traverse from one node into another maintaining their trans-node real-time characterization. In the DRTSJ, the idea of producing a real-time remote invocation is used as an underlying transport mechanism. The main role is played by the distributable thread

* Corresponding author.
E-mail addresses: pbasanta@it.uc3m.es (P. Basanta-Val), mvalls@it.uc3m.es (M. García-Valls).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.08.001

that moves its locus of execution independently from the underlying infrastructure.

In this article, the central entity is the remote invocation of RMI that is used in some RT-RMI approaches to communicate real-time Java enabled nodes. The article proposes a model for real-time remote invocations that takes into account the resources involved in the end-to-end path. The way the approach develops is starting from a typical remote invocation, extending its basic behavior with new entities that offer real-time performance. From this initial model, it is defined an extension to the current RMI and RTSJ specifications to accommodate the architectural solution that shows how to use resource for a remote invocation.

The model proposed can be used by DRTSJ to define a predictable low-level communication model for remote invocations on which it could run distributed real-time threads. The model may be also integrated in current end-to-end scheduling algorithms (like [40,59,62]) specializing their computational models; one goal that is out of the scope of the paper. Other RT-CORBA models (like those lead by RTZen [45,54]) may enhance its current architecture including specific elements defined in the model proposed (like memory-area pools).

The rest of this article is organized as follows. Section 2 is the state-of-the-art and reviews the main approaches that combine the RTSJ and RMI to produce different RT-RMI technologies. Section 3 is a background section that introduces the architecture used for developing the model proposed. Section 4 develops the model and the API based on RTSJ and RMI. The model is evaluated in Sections 5 and 6 which detail results of a set of empirical tests designed to measure the performance of the model on Java's RMI. Section 5 compares real-time remote invocations against traditional RMI invocations, while Section 6 uses real applications requirements taken from two application benchmarks. Finally, the paper ends drawing conclusions and our on-going work in Section 7.

2. Background on real-time java technologies

Due to historical reasons [41] real-time Java technology has been considered from two different angles: centralized and distributed. On the one hand, the centralized efforts for real-time Java have focused their attention on improving the predictability of single real-time virtual machines. On the other hand, the most important efforts towards a distributed real-time Java technology define mechanisms that offer end-to-end real-time performance in remote communications taking in most cases centralized technologies as departure points.

2.1. Centralized real-time java

Different approximations to real-time Java may be grouped into three categories: (i) based on modifications to Java APIs; (ii) based on integrating the virtual machine into the real-time operating system; and (iii) based on specific hardware. Among them, the most relevant for the real-time remote invocation are those based on extending current Java APIs—the main approaches to RT-RMI follow this approach. This choice constrains our analysis to the following technologies: RTSJ [5], RTCORE [6], PERC [7], RTJThreads [47], and CJThreads [42].

- *RTSJ* (The Real-Time Specification for Java) is a specification for real-time Java that defines an alternative based on modifying the virtual machine. In this specification the virtual machine supports the development of both general purpose and real-time applications. Nowadays, this specification is the most relevant in the area and several commercial products support this specification.

Table 1
Different API-based approaches for centralized real-time java.

	RTSJ	RTCORE	PERC	RTJ	CTJ
Scheduling	A	A	A	A	A
Synchronization	A	A	-	A	–
Garbage collection	A	A	A	A	–
Class loading	A	A	A	–	–
Class initialization	A	A	A	–	–
Events	A	A	A	–	–
Hardware access	A	A	A	–	A

Note: Legend: A (addressed); and – (not addressed).

- *RTCORE* (Real-Time Core Extensions) defines an alternative based on a new execution environment: the core. This execution environment offers a new class hierarchy to support the development of real-time applications which communicate with non real-time applications using queues. Some of the principles proposed by RTCORE have been recovered in the context of high-integrity systems based on RTSJ [8].
- *PERC* (Portable Executive for Reliable Control) defines two packages: one for real-time applications and another for embedded systems. Today, this product belongs to ATEGO.
- *RTJThreads* (Real-time Java Threads) is a rather simple solution; it consists only of three classes. Its scheduling model is based on priorities and real-time synchronization protocols.
- Finally *CJThreads* (Communicating Threads) is based on the CSP formalism defined by Hoare.
- Table 1 identifies the type of coverage given by each approach to key drawbacks of Java for the development of real-time and embedded systems. Among all of them, the table highlights the following: (1) lack of precise scheduling policies, (2) lack of real-time synchronization algorithms, (3) priority inversion introduced by the garbage collector, (4) the behavior of the dynamic class loader, (5) the lack of specific support for processing external events, and (6) access to low-level hardware. These limitations are described in the NIST document requirements for real-time Java [11].

2.2. Distributed real-time java

Whereas in centralized real-time Java there is a mature idea of the main problems to face, the distributed area lacks a similar consensus [16]. There are not fully developed specifications that help the programmer during the design of distributed real-time applications or implementations that may be used in the development of commercial applications. Mainly there are two promising initiatives that combine two specifications for real-time Java (RTSJ and RTCORE) with two distribution technologies: RT-CORBA and RMI.

The RTZen project [53] is one of the main approaches following the RT-CORBA path. It uses RTSJ to improve the predictability of RT-CORBA using lessons learnt from the RT-CORBA architecture. The model proposed uses resources that are defined in RT-CORBA like thread and connection pools. However, it also adds others like the memory pool that is not defined in the RT-CORBA specification. Memory area pools may be used in RT-CORBA to remove the garbage collector from the end-to-end remote invocation (following a model similar to the NhRo paradigm [22]) adapted to the RT-CORBA model. The way applications define their real-time parameterization is also different, whereas RTZen follows the RT-CORBA's architecture (i.e. with ORBs and POAs), the model proposed is closer to the RTSJ's architecture than RTZen is.

The rest of this state-of-the-art focuses on RMI-based initiatives: DRTSJ [12], RT-RMI-York [33], RT-RMI-UPM [61], and RT-RMI-TA&M [55] which are closer to the model proposed by DREQUIEMI than RTZen.