Distributed, Embedded and Real-time Java Systems

M. Teresa Higuera-Toledano • Andy J. Wellings
Editors

Distributed, Embedded and Real-time Java Systems

Springer

Editors
M. Teresa Higuera-Toledano
Universidad Complutense de Madrid
Facultad de Informática, DACYA
Calle del Profesor Gaecía
Santesmas
28040 Madrid
Spain

Andy J. Wellings
Department of Computer Science
University of York
Heslington
York
United Kingdom

ISBN 978-1-4419-8157-8 e-ISBN 978-1-4419-8158-5
DOI 10.1007/978-1-4419-8158-5
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011945437

Preface

The Java language has had an enormous impact since its introduction in the last decade of the twentieth century. Its success has been particularly strong in enterprise applications, where Java is one of the preeminent technology in use. A domain where Java is still trying to gain a foothold is that of real-time and embedded systems, be they multicore or distributed. Over the last 5 years, the supporting Java-related technologies, in this increasingly important area, have matured. Some have matured more than others. The overall goal of this book is to provide insights into some of these technologies.

Audience

This book is aimed at researchers in real-time embedded systems, particularly those who wish to understand the current state of the art in using Java in this domain. Masters students will also find useful material that founds background reading for higher degrees in embedded and real-time systems.

Structure and Content

Much of the work in real-time distributed, embedded and real-time Java has focused on the Real-time Specification for Java (RTSJ) as the underlying base technology, and consequently many of the Chapters in this book address issues with, or solve problems using, this framework.

Real-time embedded system are also themselves evolving. Distribution has always been an important consideration, but single processor nodes are rapidly being replaced by multicore platforms. This has increased the emphasis on parallel programming and has had a major impact on embedded and real-time software development. The next three chapters, therefore, explore aspects of this

issue. In Chap. 1, Wellings, Dibble and Holmes discuss the impact of multicore on the RTSJ itself and motivate why Version 1.1 of the specification will have more support for multiprocessors. Central to this support is the notion of processor affinity.

RTSJ has always remained silent on issues of distributed system as other work in the Java Community has been addressing this problem. Unfortunately, this work has been moribund for several years and is now considered inactive. It is, therefore, an appropriate time to re-evaluate the progress in this area. In Chap. 2, Basanta-Val and Anderson reviews the state of the art in distributed real-time Java technology considering both the problems and the solutions.

As embedded real-time system grow in size, they will become more complex and have to cope with a mixture of hard and soft real-time systems. Scheduling these systems, within the imposed time constraints, is a challenging problem. Of particular concern is how to integrate aperiodic events into the system. The accepted theoretical work in this area revolves around the usage of execution-time servers. In Chap. 3, Masson and Midonnet consider how these servers can be programmed in the RTSJ at the application level.

Irrespective of size, embedded systems have imposed constraints, be they power consumption, heat dissipation or weight. This results in the resources in the platform being kept to a minimum. Consequently, these resources must be carefully managed. Processors and networks are the two of the main resource types, and have been considered in Chaps. 1–3. Memory is the other main resource, and it is this that is the topic of Chap. 4 through Chap. 6. Real-time garbage collection is crucial to the success of real-time Java and the advent of multicore has added new impetus to that technology to produce parallel collectors. In Chap. 4, Siebert explains the basic concepts behind parallel collectors and reviews the approaches taken by the major RTSJ implementations.

RTSJ introduced a form of region-based memory management as an alternative to the use of the heap memory. This has been one of the most controversial features of the specification. For this reason, Higuera-Toledano, Yovine, and Garbervetsky (in Chap. 5) evaluate the role of region-based memory management in the RTSJ and look at the strengths and weaknesses of the associated RTSJ scoped memory model. The third aspect of memory management for embedded systems is how to access the underlying platform's physical memory – in order to maximize performance and interact with external devices. This has always been a difficult area and one that has been substantially revised in RTSJ Version 1.1. In Chap. 6, Dibble, Hunt and Wellings considers these low-level programming activities.

Part and parcel of developing embedded systems is deciding on the demarcation between what is in software and what is in hardware. From a Java perspective, there are two important topics: hardware support for the JVM and how applications call code that has been implemented in hardware. These two issues are addressed in Chaps. 7 and 8. In Chap. 7, Schoeberl reviews the approaches that have been taken to support the execution for Java programs. This includes important areas like support for the execution of Java byte code and garbage collection. In contrast, Whitham and

Audsley in Chap. 8, focus on the interaction between Java programs and functional accelerators implemented as coprocessors or in FPGAs.

Embedded systems are ubiquitous and increasingly control vital operations. Failure of these systems have an immediate, and sometimes catastrophic, impact on society's ability to function. Proponents of Java technology envisage a growing role for Java in the development of these high-integrity (often safety-critical) systems. A subset of Java augmented with the RTSJ, called Safety Critical Java (SCJ) is currently being developed by the Java Community Process. This work is approaching fruition and two of the most important aspects are discussed in Chaps. 9 and 10. Most embedded and real-time system directly or indirectly support the notion of a mission. In the RTSJ, this notion can be implemented but there is not direct representation. SCJ provides direct support for this concept. In Chap. 9, Nielsen and Hunt discuss the mission concept and show how it can be used. Paramount to high-integrity applications is reliability. Although Java is a strongly typed language run-time exceptions can still be generated. Recent versions of the language have added support for annotations, which essentially allow added information to be passed to the compiler, development tools supporting static analysis, and the JVM itself. In Chap. 10, Tang, Plsek and Jan Vitek explore the SCJ use of annotations to support memory safety, that is the absence of run-time exceptions resulting from violations of the SCJ memory model.

Up until now, the topics that have been addressed have been concerned with the underlying technology for real-time and embedded Java. The remainder of the book focuses on higher level issues. Real-time and embedded system designers have to be more resource-aware than the average Java programmer. Inevitably, this leads to complexities in the programming task. The RTSJ has introduced abstractions that help the programmer manage resources; however, some of these abstraction can be difficult to use (for example, scoped memory). In Chap. 11, Plsek, Loiret and Malohlava argue that this can make the use of the RTSJ error prone. They consider the role of RTSJ-aware component-oriented frameworks that can help provide a clearer separation between RTSJ-related issues and application-related issues.

One well-known Java framework for component-based service-oriented architectures is the Open Services Gateway initiative (OSGi). Although this initiative has wide support from industry, it lacks any support for real-time applications. In Chap. 12, Richardson and Wellings propose the integration of OSGi with the RTSJ to provide a real-time OSGi framework.

In the final chapter of this book, Holgado-Terriza and Videz-Aivar address the issue of building a complete embedded application that includes both software and hardware components from a reusable base.

Acknowledgements

The editors are grateful to Springer who gave us the opportunity to produce this book, and to all the authors for agreeing to contribute (and for reviewing each others chapters).

We are also indebted to Sitsofe Wheeler for his help with latex.

Andy J. Wellings
M. Teresa Higuera-Toledano

Contents