Memory Management for Real-Time Java: An Efficient Solution using Hardware Support*

TERESA HIGUERA
teresa.higuera@inria.fr

INRIA-Rosquencourt, Domaine de Voluceau, BP 105, 78153, France

MICHEL BAÑÂTRE
JEAN-PHILIPPE LESOT
FRÉDÉRIC PARAIN
GILBERT CABILLIC
INRIA-IRISA, Campus de Beaulieu, 35032 Rennes Cédex, France

Abstract. This paper addresses the issue of improving the performance of memory management for real-time Java applications, building upon the real-time specification for Java (RTSJ) from the Real-Time Java Expert Group. In a first step, a collecting dynamic memory solution including both a real-time garbage collector and region-based memory management, is proposed. A thorough analysis of the parameters influencing the performance of write barriers in memory management, together with ways of improvement are then presented. Finally, the implementation of a memory management solution compliant with the RTSJ and integrating the proposed improvements is sketched.

Keywords: Java, real-time, embedded, garbage collection, memory regions, write barriers, performance

1. Introduction

The Java environment provides attributes that make it a powerful platform to develop embedded real-time applications. However, it presents some important lacks regarding its use in this kind of systems (Higuera et al., 2000). The original Java platform was designed for computers with RAM memory (personal computers or workstations). Then, other Java platforms were defined to support real-time and embedded systems. In particular JavaCard (Sun, 1998), EmbeddedJava (Sun, 1999) and PersonalJava (Sun, 1998) are three different platforms from Sun Microsystems. But none of these Java adaptations integrate the techniques and methods that real-time systems require.

The National Institute of Standards and Technology (NIST), has produced a basic requirements document (Carnahan, 1998) for a standard real-time Java API extension. Solutions that comply with this document are the real-time specification for Java (RTSJ) (RTJEG, 2000) and the real-time core extension for the Java platform (RT Core) (J Consortium, 1999). There are some other solutions that were introduced before the NIST document. The simplest one is a prototype introducing tasks support over RT-Mach (RT-Threads) (Miyoshi et al., 1997). Another proposal is the portable executive for reliable

* This work has been partially funded by Texas Instruments.
control (PERC) (Nilsen, 1998), that is close to the J Consortium solution, the latter being actually an evolution of PERC. A very different solution is the communication threads for Java (CTJ) (Hilderink, 1998), that is based on the CSP algebra, the Occam2 language and the Transputer microprocessor. Other solutions integrate the JVM into the operating system such as the GVM (Bak et al., 1998), a prototype centered around resource management. Another option to improve the performance of Java is to integrate the JVM in a microprocessor as the Aj-100 (Hardin, 2001), which implements the entire JVM instruction set in silicon and directly supports the Java thread model in hardware.

We have analyzed and studied how the above solutions resolve the problems that Java presents to effectively support embedded real-time applications (Higuera et al., 2000). We have divided these problems in the following categories: (i) the inability to access the underlying hardware, (ii) the unspecified behavior for thread scheduling, (iii) synchronization that requires stronger semantics, (iv) the inability to handle events, (v) the inability to specify resources, and (vi) dynamic memory management. A comparison of the aforementioned solutions is summarized in Table 1.

From our point of view, the RTSJ constitutes the most adequate solution for real-time systems in general, and a hardware support such as Aj-100 enables improving the system’s performance. In the context of the activities of the Solidor group at INRIA, we are currently developing a Java-based software environment accounting for embedded real-time (Issarny et al., 2000). This paper focuses on how to make Java memory management real-time while accounting for relevant Java specifications: the RTSJ, the KVM (Sun, 1999) targeting limited-resource and network connected devices, and the picoJava microprocessor core (Sun, 1999).

### 1.1. Background

Implicit garbage collection has always been recognized as a beneficial support from the standpoint of promoting the development of robust programs. However, this comes along with overhead regarding both execution time and memory consumption, which makes (implicit) garbage collection poorly suited for small-sized embedded real-time systems. This must not lead to undertake the unsafe primitive solution that consists in letting the

---

**Table 1. Comparison of studied solutions.**

<table>
<thead>
<tr>
<th></th>
<th>(i)</th>
<th>(ii)</th>
<th>(iii)</th>
<th>(iv)</th>
<th>(v)</th>
<th>(vi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTSJ</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
</tr>
<tr>
<td>RT Core</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>M</td>
<td>A</td>
</tr>
<tr>
<td>PERC</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>M</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>RT-Threads</td>
<td>A</td>
<td>A</td>
<td>—</td>
<td>—</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>CTJ</td>
<td>A</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>GVM</td>
<td>M</td>
<td>A</td>
<td>—</td>
<td>—</td>
<td>M</td>
<td>—</td>
</tr>
<tr>
<td>Aj-100</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

*Notes:* We use A, M, and—to respectively mean that the corresponding issue is addressed in detail, only partly addressed, and not addressed.