

A Java Embedded Micro-kernel Infrastructure

Mark Baker and Hong Ong

Distributed Systems Group

University of Portsmouth

Portsmouth, PO1 2EG, UK

Mark.Baker@computer.org and Hong.Ong@port.ac.uk

Categories and Subject Descriptors

D.3.4 [Programming Languages]: Processors – Run-time environments, Memory management, Optimization, Interpreters.

General Terms

Design, Languages, Performance, Management

Keywords

Java, JNI, Micro-kernel, Modules, Communication API

This abstract summarizes the key ideas of the Java Embedded Micro-kernel (JEM) infrastructure that is capable of dynamically loading service modules at runtime onto devices with limited system resources.

Our initial motivation for undertaking this project was driven by several needs:

- An ongoing project, MPJ, and its needs for fast and reliable communication.
- The ability to move services from system and user-space down into the network layer, for example, packet filtering or daemon services.
- The ability to test the design and implementation of services for embedded devices.

In particular, these application services would by-pass the normal performance bottlenecks found when implemented as system or user level services. Our motivation has led us to believe that we need to develop and implement a generic JEM infrastructure that can be used in a range of embedded devices - from network cards to PDAs and from mobile phones to printers.

Central to this infrastructure is a Java-based micro-kernel, which will provide the necessary mechanisms for the various runtime aspects of the system. In addition, the micro-kernel will allow a range of service modules, written in Java, to be dynamically loaded at runtime. The micro-kernel together with the service modules forms the Java Embedded Micro-kernel (JEM) infrastructure.

Figure 1 shows the major components in the JEM architecture.

- The Abstract Hardware Layer (AHL) defines “the red line” between Java and hardware dependent code. It is the means

by which JEM interacts with the underlying device hardware such as the processor, registers and memory.

- The JEM JNI API is the fast by-pass route for the JEM communications API to interact with the device AHL.
- The JEM communications API provides a messaging interface to JEM modules that require data I/O services. This API interacts with its peer on other devices.
- The JEM modules provide application level services for a Java application.
- The Java Embedded Micro-kernel provides the operating environments for the JEM modules and manages system resources and their usage.

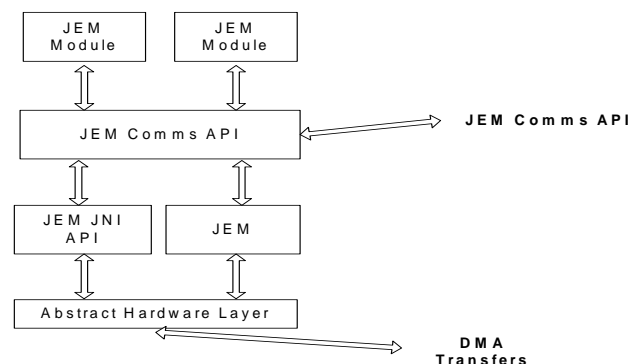


Figure 1: Generic JEM Architecture

The design of JEM is based largely, but not entirely, on L4 system that was originally proposed by Liedtke in the mid 90's. However, our implementation is being carried out in Java as opposed to the original assembly language and C version.

JEM supports three abstractions – modules and threads, address space, and messages. A module contains the resources associated with a process and does not perform computation itself but serves as a framework in which thread(s) can operate. A thread contains the minimal processing state associated with a computation. Address space is a mapping associated with each virtual page to a physical page frame. Each module has a virtual address space within which its thread executes. Like L4, JEM supports the recursive construction of address space concept. All inter and intra communications among modules is carry out in the form of messages. A message is considered as a stream of bytes and contains the data to be communicated

For a full-length version of this paper and related works, see <http://homer.csm.port.ac.uk/projects/research/jem/>

COPYRIGHT IS HELD BY THE AUTHOR/OWNER (S).

JGI'02, NOVEMBER 3–5, 2002, SEATTLE, WASHINGTON, USA.

ACM 1-58113-599-8/02/0011.