

A Proposed Initial Approach to Distributed Real-Time Java: Extensions to RMI for End-to-End Timeliness QoS

E. Douglas Jensen

The MITRE Corporation

jensen@real-time.org, <http://www.real-time.org>

Extended Abstract

The “Real-Time Specification for Java” is being written by the Java Specification Request JSR-1 Expert Group (of which this author is a member) as part of Sun’s Java Community Process, and will reach version 1.0 by September 2000. That specification is being extended to the “Distributed Real-Time Specification for Java” by the JSR-50 Expert Group (which this author leads). This paper is about the proposed initial approach to distributed real-time Java, as described in JSR-50. The approach is based on enhancing Remote Method Invocation (RMI) to support end-to-end Quality of Service (QoS) – particularly including, but not limited to, timeliness. The enhancement mechanism allows RMI to transparently acquire appropriate context (for example, the invoker’s time constraint and other scheduling parameters) from the invoking node OS/JVM/etc., propagate it to the invoked node, and deposit it in the corresponding place(s) there. Thus (for example), the invoked execution can be scheduled with the same eligibility (e.g., priority, deadline, etc.) as the invoking execution. This mechanism supports a wide variety of distributed programming models requiring end-to-end QoS, including the *activity* model in the proposed Dynamic Scheduling Real-Time CORBA (and elsewhere).

The JSR-50 proposal takes the position that “distributed real-time Java” would initially most usefully be focused on supporting control flow style distributed object programming models. Such models are better understood and more widely accepted in the real-time applications communities than are more Java oriented ones, such as mobile objects. Other styles of real-time distributed programming models, including data flow (e.g., publish/subscribe) and mobile objects, should be the topic of subsequent JSR’s.

More specifically, control flow in this approach means a program can be distributed in the sense that one or more of its computational entities can execute across physical node boundaries, instead of being confined to a single node or a single object instance on a node, as a conventional thread is. In this paper, we will use the term *activity* for such a *trans-node* computational entity in a distributed program. An activity can extend and retract its locus of execution point movement among a program’s constituent methods in object instances that may be dispersed across a multiplicity of physical computing nodes, by method invocations and returns. Within an object instance, the flow of control is normal thread execution. There may be

multiple concurrent activities in a distributed program.

In Java, an activity would be implemented by the concatenation of local (per node) threads sequentially performing RMI’s and (usually) RETURN’s.

Each activity may have one or more execution scheduling attributes – e.g., priority, time constraints such as deadlines or utility functions, importance – that specify the acceptable end-to-end timeliness for it completing the sequential execution of methods in object instances that may reside on multiple physical nodes. The semantics of acceptability with respect to these end-to-end timeliness attributes is defined by the application, in the context of the scheduling policy being used. Execution of the activity is governed by those scheduling parameters, regardless of the activity’s execution point transiting nodes.

The most fundamental requirement for this distributed real-time Java approach is that there be end-to-end transitivity of an activity’s timeliness properties – time constraints, expected execution time, execution time received thus far, etc. – along the locus of its execution point movement among local and remote object instances. These properties must be propagated between corresponding Java virtual machines (JVM’s) with the activity’s execution point movement by RMI’s and (usually) RETURN’s, and used by the scheduler at each of those nodes.

JSR-50 takes the point of view that the enhancements needed to preserve the end-to-end timeliness semantics of an activity are the essence of “real-time RMI.” Thus, “real-time RMI” doesn’t necessarily mean “real fast RMI” or even “real predictable RMI,” as in conventional real-time thinking, although both of those are important attributes. Instead, JSR-50’s point of view is an end-to-end QoS one.

The activity approach to control-flow style distributed programming model is applicable to real-time systems, which are hard or anywhere else on the predictability continuum. That continuum is orthogonal to application time-frame magnitudes, which range in practice from microseconds to megaseconds. Both time-frame magnitudes and predictability are dimensions of timeliness QoS.

The activity entity as described here is based on the *distributed thread*, and associated programming model, abstractions in Jensen’s Alpha distributed real-time OS kernel at CMU, and much subsequent development and experience by the author and others.