Report of Session:
Analysis of the J Consortium Real-Time Java Proposal

chair: Brian Dobbing

rapporteur: Tullio Vardanega

As a ‘first’ in the history of IRTAW, the workshop devoted a full session to the analysis of the real-time extensions currently proposed for a mainstream language in direct competition with Ada, that is Java. The objective of the session was to offer the group the opportunity to reflect on the status and rationale of the latest definition of the real-time extensions promulgated by the J Consortium. A subsequent IRTAW session addressed the analysis of the real-time extensions defined by the Real-Time Expert Group led by Sun Microsystems.

The session chair, an active member of the J Consortium technical committee, presented a short overview of the purpose, the structure and the key features of the definition put forward by the J Consortium. Refer to Brian Dobbing’s paper in the proceedings of IRTAW 2000 for a more elaborate account of those aspects.

In the following we highlight the key issues that emerged from the discussion that ensued from the chair’s presentation. Issues are recorded in the chronological order of discussion, not necessarily reflecting their relative importance.

The J Consortium follows an open, consensus-based process aimed at the submission of an ISO Publicly Available Specification (PAS). The IRTAW group appreciated the intent but expressed concern at the fact that the J Consortium PAS would not be able to base on a public standard and would thus be exposed to the risk of impacting changes in relevant elements of the Java proprietary specification.

The J Consortium has opted for a tighter interpretation of the real-time requirements than demanded by the NIST requirements document. This approach warrants better resource performance and execution predictability than otherwise achievable, yet at the cost of reducing inter-operation with normal Java. In keeping with this objective, the J Consortium proposal (collectively referred to as Real-Time Core Extensions) envisages real-time Java programs to execute on a Core engine, independently of the normal JVM, whilst allowing those with less stringent integrity and predictability requirements to use typical JVM services such as dynamic load and APIs via controlled interaction with the JVM.

The IRTAW group noted that the J Consortium proposal entails a true extension of Java, as opposed to a revision, where the latter option was in fact ruled out by the lack of a public standard to base on.

As a reflection of this strategy, programs written for execution on the Real-Time Core would not be able to run on the normal JVM. The vast majority of current Java APIs would thus have to be rewritten for execution on the Real-Time Core and, more generally, for use in hard real-time high-integrity applications. In fact, the J Consortium operates on the assumption that very few APIs would be directly needed within Core programs, so that only those would have to be rewritten to be Core-compliant. The vast number of existing APIs would continue to be used unchanged for JVM execution, in parallel to and on controlled interaction with Core execution.

The discussion then moved on to address the fundamentals of the real-time model promoted by the J Consortium and to draw parallels with the corresponding Ada (and Ravenscar) definition. The IRTAW group looked with interest at the flexibility with which users of the Real-Time Core are able to map activation events to threads. The group also noticed however the cumbersome syntax presently required to create periodic threads at program level.

The group observed that the J Consortium specification demands a significantly larger priority range ($\geq$128) than required by the ALRM ($\geq$32). This choice may complicate the binding of the Real-Time Core to POSIX platforms. The chair observed however that Real-Time Core threads need not necessarily be implemented in terms of underlying RTOS threads, if any. The group also noted that the J Consortium specification does not guarantee a minimum number of software priorities, while the ALRM requires at least 31 of them.

The J Consortium specification includes the Ada-like notion of (entry-less) protected object. The Real-Time Core specifies priority inheritance (PI) as the default protocol for

the reduction of priority inversion on access to protected methods. Protected objects may alternatively choose priority ceiling (PC). As a consequence of this optionality, however, PI and PC protocols may coexist on different (views of) methods in the one and the same class hierarchy. In fact, the chair clarified that J Consortium prescribes the use of the PC protocol only for all synchronised methods in high-integrity systems.

In contrast with Ada, the Real-Time Core supports the implementation of dynamic ceilings, and a fairly straight-forward one at that. Also contrary to Ada, protected (i.e.: synchronised) regions are not abort deferred by default. They become so only as their interface is extended to Atomic.

The IRTAW group looked with interest at how the J Consortium defines asynchronous transfer of control (ATC) events and the relevant handling mechanisms. Drawing a parallel with the corresponding Ada notion, the group noted that where Ada specifies abort deferral as the required ATC behaviour and declares abortable regions explicitly, the Real-Time Core makes abort-deferred behaviour optional (thus explicit) and all operations in principle abortable. The net consequence of notion is that the use of any legacy Java code in a Core ATC construct would neither be abort-deferred, since abort deferral must be explicitly programmed, nor generally useful, since the legacy code itself might not be written to be abortable.

Overall, the group regarded the J Consortium specification as a worthwhile engineering effort and enjoyed the opportunity to analyse the J Consortium approach (with all its Java flavour) to the resolution of real-time issues long known to the IRTAW community.

Overview of the Sun Java Community Process's Real-Time Expert Group specification of RT-Java Session Summary

Brian Dobbing
Aonix Europe Ltd,
Partridge House, Newtown Road,
Henley-on-Thames, Oxon RG9 1EN, United Kingdom
brian@aonix.co.uk

1. Introduction

The session was chaired by Ben Brosgol of Ada Core Technologies.

Ben presented an overview of the main points of the specification. He pointed out that during the production of the reference implementation and conformance test suite, further modifications are possible and that the final definition would include all three components (specification, reference implementation and test suite). Andy Wellings presented some comments on the scheduling support offered by the Sun specification.

2. Real-Time Expert Group Specification

2.1. Concurrency

The specification emphasizes predictability and a flexible scheduling framework over performance, in contrast to the J Consortium approach. The threads model is designed to operate predictably even in the presence of a garbage collector. Periodic and aperiodic threads are defined in addition to interrupt handlers.

2.2. Memory Management

New memory areas are defined; immortal memory and scoped memory; to supplement the garbage collected heap. Assignment rules checked at runtime prevent dangling references in the scoped memory. Raw (physical) memory access is also defined.

2.3. Asynchrony

Asynchronous event handlers are defined and have thread-like behavior. There can be a many-to-many mapping between events and handlers.

Asynchronous Transfer of Control (ATC) is supported. In contrast to the J Consortium approach, ATC cannot have

resumptive semantics, only termination semantics; all synchronized code is abort-deferred but finally clauses are not; ATC-susceptible regions must be explicitly identified by a specific *throws* clause in the signature, and hence all existing Java code is abort-deferred by default.

2.4. Scheduling

FIFO_Within_Priorities base scheduler must be supported but not necessarily as the default.

At least 28 distinct priority levels are required for RT threads in contrast to the 128 required by the J Consortium specification.

2.5. Locking

Priority inheritance and Priority Ceiling Protocol (PCP) are both defined. Priority inheritance is the principal one used for synchronized code. There is no requirement to report blocking operations or ceiling violations as an error in PCP, although this is an oversight that may be corrected in a revised version of the spec. Dynamic ceilings may be achieved programmatically.

2.6. Dynamic Scheduling Analysis

CPU time budgeting and on-line scheduling feasibility analysis is defined but optional. Changes in thread scheduling parameters are not synchronized automatically; this must be achieved programmatically.

An issue for sporadic threads was identified in the assumption that the deadline equals the minimum interarrival time. Thus it was not clear how to detect overrun of deadlines for real-time threads with sporadic release parameters.

There are no examples of how to implement sporadic and deferrable servers. Also, one can group threads together with respect to cumulative CPU time usage and budget, but it is not clear how this can be used. Textbooks are in progress to provide examples of intended usage of the new features.

Many models exist for dynamic scheduling analysis, but only one is supported by the model. The workshop view is that as long as feasibility analysis is available, the features defined are sufficient given the additional hooks that are also provided.

3. Final Comments

The latest public version of the J Consortium specification (1.0.10) suffers from being a very poorly presented document, with much non-normative text, repetition of topics, difficulty in locating the API definition, and some internal contradictions. These presentation problems need to be addressed prior to submission for ISO standardization. In contrast, the Sun specification is well presented.

The workshop concluded that both specifications are worthy engineering achievements.