Session Summary:

Integration versus Orthogonality (RTSJ scheduling policies versus Ada's)

Chair: Andy Wellings
Rapporteur: Joyce L. Tokar

Abstract

The purpose of this session was to review the Real-Time Specification for Java (RTSJ) and look at the scheduling that Java offers. The discussion focused on the assertion that Java offers a higher-level of abstraction for scheduling than Ada.

1 Introduction

One aspect of the current work being done in the Real-Time Java community is focused on developing an integrated high-level framework for real-time programming. The premise of this session is that the integrated approach of RTSJ is superior to the low-level model offered in Ada. The goal was to stimulate a lively debate in the strengths and weaknesses of the two approaches.

2 RTSJ Background

Java 1.5 is expected to be available at the end of 2003. This version of the language introduces generics, enumeration types, and predefined concurrency utilities. There are some efforts underway to investigate safety-critical Java as well. There is an interest in getting the safety-critical Java definition standardized and made more open to the public with respect to copyrights and licensing.

There are a number of requirements that are to be met by the RTSJ including:

- NIST requirements
- Memory management
- Time values and clocks
- Schedulable objects and scheduling
- Real-time threads
- Asynchronous events and timers
- Asynchronous transfer of control
- Synchronization and resource sharing
- Physical memory access

While the IRTAW has considered asynchronous transfer of control (ATC) for Ada, we have never considered the scheduling approach that is being proposed for RTSJ.

3 RTSJ – Scheduling Approach

The chair summarised the RTSJ model. The following contains the main points of that summary.

In theory, RTSJ allows one or more schedulers to exist within a given RTSJ environment. Each schedulable object is assigned a scheduler. All schedulers provide a common interface.

In practice, priority based scheduling is well supported. However, there are probably not enough hooks to support other schedulers at this time. Presently, no real consideration has been given to the implications of multiple schedulers. Future development is expected to address these issues. The current focus is on developing an integrated model to support priority based scheduling, cost enforcement, and deadline monitoring.

There are several questions that need to be addressed with respect to scheduling policies in Java including:

- What is the release profile of the thread?
- What is the cost per release?
- What is the execution deadline?

To address these concerns each schedulable object in RTSJ is characterized (per mode) by its:

- Release profile: periodic, aperiodic, sporadic,
- Processing cost per release
- Deadline
- Value
- Other hardware resources needed per release (e.g. network capacity)
- Software resources per release.

The scheduler/application must ensure that any deviation from a schedulable object's characteristics does not undermine any schedulability analysis that has been performed (off-line or on-line). The scheduler/application must provide mechanisms that can bind the impact of schedulable objects that have non-periodic release profiles.

There was a question from the workshop regarding support for the dynamic building of processes. RTSJ does offer a framework for this but the hooks are not always sufficient.

The RTSJ model indicates that the virtual machine (VM) controls the release of schedulable entities as follows:

Periodics both time triggered and event triggered

Sporadic – various approaches to minimum inter-arrival violation are supported by the RT JVM

Aperiodics – a capacity-sharing scheme is supported which ensures groups of schedulable entities can consume no more that a certain capacity

In practice, the support for periodic time trigger events is fairly well supported. However, periodic event triggered releases are still not handled very well at this time. Similarly, sporadic and aperiodic real-time threads are not well supported.

The discussion continued with a request for an example of an event triggered periodic. An example is a clock device. The common practice in the RTSJ community is to use real-time threads for periodic events and event handlers for event driven threads.

3.1 CPU Budgeting

There are no metrics of the accuracy of the CPU Budgeting. The implied type is a nanosecond. There is no requirement to document the Virtual Machine (VM) overhead.

RTSJ is not nearly as well defined as the Real-Time annex. The underlying model is that the scheduler will only give you the CPU budget requested unless otherwise specified. If you have a handler for overrun of time, then the handler can give the thread more CPU time. This is quite a strong model as the scheduler controls the amount of time that thread may have. The scheduler monitors the cost per release. It cannot monitor a cost on an operation as budgeting is done at the schedulable object level. The program cannot find out how much budget it has left. There is nothing in the RTSJ to determine how much of the schedule has been used.

CPU Budgeting is well supported for periodic threads. Not well defined for aperiodics.

3.2 Deadline Monitoring

Missed deadlines are reported in an overrun handler if you have one, otherwise, reported at the end of your current release.

There is no direct support for mode changes -- can do this with a change of scheduler.

3.3 Hardware Resources

Can specify your memory requirements then the RTSJ VM monitors your memory usage. Presently, there really are not enough hooks to respond to error conditions.

3.4 Software Resources

There is support for priority inheritance and ceiling priorities. Support for blocking time is missing.

4 Ada vs. Java Discussion

The debate on the strengths and weaknesses of the Java and Ada scheduling models was wide ranging. The following captures the main points.

The RTSJ scheduler needs to know what the real-time overhead is for the RTSJ VM; the programmer does not need to know. The RTSJ is a dynamic world; the framework is provided for different concurrency models and different schedulers. Presently, RTSJ does not have enough hooks for application level schedulers or system level schedulers. Nor does it have enough hooks for multiple schedulers right now.

In RTSJ, everything is extensible. The scheduler provides essentially a yes/no to the application on whether it is schedulable. You get the scheduler of the implementation not application level defined. Therefore, a piece of the Java VM needs to be recreated for each piece of hardware on which it is going to run. Thus, you cannot write an application level scheduler to replace the default scheduler without knowledge of the hardware.

Some members of the IRTAW think the Java model is too complex. The default framework appears to be rather complex and does not seem to meet many of the real-time requirements.

In RTSJ, there is a shifting of complexity from the application to the scheduler that is controlling when your application is released. RTSJ is prescriptive on the things that are needed for scheduling. Outside of this, the application can handle its own behaviour.

In RTSJ, how do I run a thread as frequently as possible? How much information can the scheduler provide?

If you want to define a scheduler other than what is the default scheduler, then extend the default class. In addition, you may disable the features of the framework that you are not going to use. Many of the aspects of the default scheduler are optional.

What are the hooks to build an application level scheduler? The default scheduler lacks the features that are really needed in a real-time application.

RTSJ is not a mature technology; support for application level schedulers is still being developed.

The periodic task abstraction was found to be too restrictive in Ada. The `delay until` statement now lets you build what you want.

RTSJ gives you templates to build the things you want. Presently, RTSJ is missing a way to communicate blocking time to the scheduler. Event handler provides the scheduling characteristics in a framework so you could provide something other than a thread. The model provides the hooks. TimeSys has an implementation on top of real-time Linux right now.

Ada has a strange combination of low-level and high-level mechanisms. Can we implement the Java model in Ada? Do we need to extend the Ada budget time model for groups of task rather than the per task model?

Juan Antonio has done some work on patterns and frameworks for common problems. Really need to publish, in a public forum, a collection of these patterns.

Ada does not generate the secondary standard items like the Java and C# communities provide. This lack of supporting software models slows down the use and acceptance of Ada.

Do we have enough low-level mechanisms in Ada? What is stopping the production of Ada packages such as multiple schedulers from being developed? The hope is that the workshop will generate the AIs necessary to meet the requirements here. Not suggesting any new major features for Ada rather adding slight modifications to Ada to have the benefits of RTSJ in Ada. Can we define the information and constructs necessary to build the paradigms in the scheduling area that are similar to RTSJ?

We have come to the point where we believe that the focus here is to build the hooks and proposals for a variety of scheduling models in Ada. We also need to address resource utilization and networking.

The user community has not accepted the Ada model for distribution that is defined in the Distribution Annex. There was some discussion on the alternatives that are available in Ada for distribution.

The discussion moved into the domain of space and time partitioning. Space and time partitioning is a very cost effective way of handling verification and validation. It would be nice to use the concept of a partition without the Distribution Annex. Need interpartition communication and interpartition scheduling.

We can use shared passive partitioning and storage pools for spatial partitioning and data sharing. We still have an issue with interpartition communication.

Is there any emphasis in Java for space partitioning? There is some work on isolation of application issues but

this has not made its way into RTSJ. May be able to get this into the safety critical RTSJ.

5 Conclusions

RTSJ is developing a model that provides high-level mechanisms for real-time programming within an overall approach. As this model is under development, the actual implementation is immature. The real-time community as well as the safety-critical community are contributing to the overall definition of RTSJ.

Ada on the other hand offers a number of low-level mechanisms to enable the construction of a variety of scheduling paradigms. The IRTAW is challenged to find the best solutions for Ada2005.