Session 2: Programming Patterns and Libraries

Chair: Michael González Harbour
Rapporteur: J. Javier Gutiérrez

1. Session Goals

The chairman introduced the session pointing out its main goals, which were to:

- discuss different visions of architectural frameworks and coding patterns with Ada 2005
- and formulate proposals and roadmaps to facilitate adoption or secondary standards.

Afterwards, the three papers [1][2][3] included in this session were briefly introduced by their authors followed by the discussion on the issues proposed in them.

2. Real-Time Utilities

Andy Wellings introduced the topic included in paper [1] by showing how real-time utilities can be provided as extensions to Ada, looking at previous experiences coming for example from Real-Time Java, which has a lot of available utilities. It was stated that, from the beginning, Ada 83 provided good and relatively simple low-level mechanisms for programming those kinds of systems, but considering the complexity of the new mechanisms introduced in the Ada 2005 and the requirements of modern applications it is desirable to provide higher-level utilities. Modern programming languages follow this path so there is a need for having this support also in Ada, perhaps through a secondary standard or some other dissemination process.

Once the initial position was set, Andy talked about the following issues that an application may require: templates (for periodic, sporadic or aperiodic tasks), to limit the amount of CPU-time that a task can receive by its association to an execution-time server, what happens in the case of a deadline miss or an execution time overrun, and the nature of the mode changes.

The proposed approach splits the support for the task into four components, each one encapsulated in a package: the application functionality (`Real_Time_Task_State` package), the mechanisms needed to control the release of real-time tasks and to detect the deadline missed or execution time overruns (`Release_Mechanisms` package), the response of tasks to deadline misses or execution time overruns (`Real_Time_Tasks` package), and the mechanisms to ensure that a subsystem is only given a certain amount of the CPU resource (`Execution_Server` package).

The specification of the `Real_Time_Task_State` package defines the `Task_State` type as the root of a class with operations to execute on each release of the task and when a deadline miss or an execution time overrun occurs. Then the inherited classes for periodic, sporadic and aperiodic tasks can be derived (see the position paper [1] for details). The main idea is that this structure is provided to the application developer as a ready-to-use infrastructure so that he or she can just concentrate on the functionality. An initial comment was made to propose the usage of interfaces to implement the state of a task. The difficulty of having public data attributes in the specification could be solved by adding set/get operations to manage these attributes.

The implementation of the release mechanisms specifies every release mechanism as a synchronized interface derived from the root type and separated in child packages for only deadline miss notification, only overrun detection, or both. The execution time servers were briefly introduced as the mechanisms capable of guaranteeing the usage of a fixed amount of the CPU resource for periodic, aperiodic or sporadic tasks. Finally, the action tasks for this framework are implemented via task templates; an example of a real-time task with deadline termination was shown. This finished the presentation of the position paper and led to an initial discussion.

The following topics were pointed out as part of this discussion:

- A question about whether it is possible or not to pass some data on the release.
- The possibility of addressing mode changes. Information on the current mode might be included in the release code, or could be added as an additional parameter to the operation `Inform_Of_a_Deadline_Miss`, although

this could allow extensions in ways other than expected and become more complex.

- The possibility of having several kinds of notification. It was stated that with the individual handling of notifications and their possible combinations, adding more kinds in the future would make the framework become much more complex.
- A question on whether a pattern such as x over y times of deadline misses can be constructed into this framework.
- The possibility of having a matrix of pending notifications or some hierarchy on the notification treatment.

At this point it was suggested to go to other presentations and come back later to a more general discussion.

3. Programming Servers in Ada 2005

Alan Burns presented the second position paper [2], which deals with the issue of programming execution-time servers in Ada 2005. He introduced the new features of Ada 2005 to control the execution time of a group of tasks managed together and to stop all of them if some budget is exhausted, and showed an example for a deferrable server running all client tasks (previously registered) at the same fixed priority.

Afterwards, he set the aim of their proposal to classify different types of servers that programmers might wish to use, in order to produce a library of these servers. The types of server should be classified according to the following characteristics:

- Dispatching: all clients have the same priority (serial), servers have a unique range of priorities (concurrent), or each client has a priority (similar to the Java model with processing groups, which is difficult to analyze). A comment was made to take into account the possibility of having only one client per server.
- Scheduling: Fixed Priorities, EDF, and perhaps mixed schemes.
- Replenishment: periodic, or related to usage (limiting or not the level of concurrency).
- Identifying sessions: to know if a client is active or not in the server, or to notify the server when a client is active, when it blocks, and when it ends its execution for the current instance. It should be useful for soft real-time.
- Whether a client should be informed or not when the budget is exhausted.
- Client binding: whether the client should follow the behaviour of the server or not.
- Different models for capacity sharing.

As a conclusion, the motivation is to see if Ada 2005 is useful for programming these issues, and to identify different server types in addition to deferrable, sporadic, periodic, constant bandwidth preserving, etc.

Once the presentation was finished, it was asked if the servers could be integrated into the previously discussed framework. The answer was that it could be addressed once the mechanism for session control is defined and the details of its integration with the framework are solved. It was suggested that priority bands could emulate a server by attaching the task to a band. Finally, it was expressed that the main idea is to put together different types of dispatching and scheduling policies, and that the same reasons that move people to use Ada, could move them to use the frameworks.

4. Ada 2005 Code Patterns for MDA

The third position paper [3] was presented by Tullio Vardanega, who talked about a 4-view MDA (Model Driven Architecture) design space. The notion is quite similar to that presented in [1], i.e., to propose templates and a framework that programmers can follow. This framework has been designed to support the usage of Ada in applications oriented to the Ravenscar profile. The main goal is to allow the designer of an application to concentrate on its functional parts, i.e, the designer only needs to worry about the sequential parts of the code, and not about other aspects like concurrency for example.

The framework gives an interface of a model to build the application. This is made via the Application-Level Containers which hold the functional modeling of the application. These containers are not executable but they are all that the designer needs to work with. The framework has other types of containers, called Virtual Machine-Level Containers, into which the Application-Level ones are mapped, and that support the execution aspects of the framework. The idea is to show if the whole process is correct, i.e., if the original structure corresponds to the final application.

Tullio showed an example about how concurrency can be overlaid. In this example the designer only has to write an operation corresponding to the functional view of the framework, while the rest of the views are offered as templates where nothing has to be coded. This way, the operation is a method inserted into a complex framework (very similar to a components framework). Again, the main idea is not to involve the user in the structural Ravenscar or real-time part. The proposed framework is currently working on an UML and Eclipse environment, fully automated and producing Ravenscar code. Another idea is to explore the use of Ada interfaces instead of generics.

5. Discussion

Following the introduction of the three position papers made by their authors, a main discussion started with an initial question by the chair of the session related to how to proceed, i.e., if some kind of standard should be produced. The idea is to produce a unique framework probably proposed as a secondary standard. The comments that this question arisen can be summarized as follows:

- The question is whether or not we can do it and how it can be funded.
- A comment to focus on paradigms and not only on the language.
- To find funding for the different developments, it would be a good idea to have an initial comparison, looking at similar efforts such as those by the companies exploring Java. It was suggested that this would be a second step and before having a full implementation, people could contribute by implementing parts of the framework (as student projects or things like that).
- It was expressed that it was early to propose the frameworks as a secondary standard.
- It was asked if anybody was familiar with the Container Library, and the process by which it had been standardized. We could follow the same path as a second step, and perhaps ARTIST could give support to the first step.
- It was suggested to use the framework proposed in [1], perhaps borrowing ideas from the framework proposed in [3], and trying to integrate the servers presented in [2].
- It was suggested there was a need for minimal documentation in order for other people to contribute to the project with new servers or components, and to be familiar with the framework.

The discussion then focused on Andy's slides to dwelve deeper into the use of an interface for the definition of the `Task_State` in the `Real_Time_Task_State` package instead of an object with attributes and methods. The issue brought up a lot of comments and questions that can be summarized as follows:

- A reason to have an interface is to be able to add other concepts in the future, for example, fault tolerance.
- It was asked if `Any_Task_State` could be an access to an interface.
- An alternative was proposed to do the type `Task_State` a tagged private type and provide get and set methods for the attributes.
- It was suggested to add a `Task_ID` attribute to the state, with only a `get` operation.
- It was discussed where to put the initial priority. Suggestions to put it in the creation of the task, or to pass it as a discriminant (with the deadline) in order to avoid `set` operations (for initial priority and initial relative deadline). It was stated that having dynamic priorities is more useful, so finally `set` and `get` methods are needed.
- It was addressed what happened if a change on the relative deadline occurs, and when does it take effect. It was suggested to be at the next release of the task.
- Mode changes were also discussed with the meaning of having a task executing a different code for each mode and with different parameters (deadline, priority, etc.) for each mode. Solutions proposed were: derive a root type from `Task_State` with a collection of modes, or an array of `Task_State` so as to be able to change from periodic to sporadic for example, or allow a different task for each mode. The mode change issue is by itself very complex, so it was decided to focus the discussion on other issues, and leave the mode change aside, as future work.

At that point, the discussion turned onto the release mechanisms proposed in the framework [1]. The release mechanism is based on two operations `Wait_For_Next_Release` and `Inform_Of` (a deadline miss, an overrun, or both), and it is thought to build real-time task patterns using the select-then-abort statement. It was suggested to create a general abstract mechanism for notification. After a long discussion on that issue, it was agreed to implement the notification object (which consists of a set of events) with a synchronized interface as follows:

```
type Notification_Object
     is synchronized interface;
procedure Add
     (N: Notification_Object;
      E: Event_Ptr) is abstract;
procedure Trigger
     (N: Notification_Object;
      E: Event_Ptr) is abstract;
procedure Wait
     (N: Notification_Object;
      E: out Event_Ptr) is abstract;
-- Wait should be implemented with an entry
-- A function to delete events is also
-- necessary
```

where the event is implemented as a tagged type without operations:

```
type Event_Kind is tagged with null record;
type Event_Ptr is access Event_Kind'class;
```

So it is possible to create deadline miss, overrun or other events by extending `Event_Kind`.

It was discussed whether the parameter for `Wait` should be an array of `Event_Ptr`, but it was decided that it was simpler to notify a single event each time `Wait` was called.

Another issue is how often the set of events to be notified is changed. If the set changes often, the `Add` operation would be eliminated and the set of events would be passed to the `Wait` operation at each call. It was considered however that the set of events is rather static and therefore separating the `Add` operation from the `Wait` operation is more convenient.

In addition, it would be necessary for the release mechanism to create an operation to add a reference to a notification object.

After finishing the discussion on the framework [1] and coming back to the framework proposed for Ravenscar [3], it was identified that the first one did not have the ability to pass parameters to sporadic tasks while the latter had. Tullio was asked to contribute to the framework [1] by integrating this ability.

6. Conclusions

The main conclusion is that some progress had been made but much more work is needed. The chair of the session suggested to propose a work plan:

- Alan Burns proposed to address the collaborative work needed to enhance a common framework via ARTIST meetings.
- Andy Wellings proposed some specific plans:
  - To change the notification object.
  - To work a little bit more on the framework before organizing a meeting to show the progress.
  - To ask for cooperation in specific topics of the implementation.

References

[1] Wellings, A.J., Burns, A.: *A Framework for Real-Time Utilities for Ada 2005*. Ada-Letters (this issue).

[2] Burns, A., Wellings, A.J.: *Programming Execution-Time Servers in Ada 2005*. Ada-Letters (this issue).

[3] Pulido, J., de la Puente, J.A., Bordin, M., Vardanega, T., Hugues, J.: *Ada 2005 Code Patterns for Metamodel-Based Code Generation*. Ada-Letters (this issue).