European Space Agency Contract Report
The work described in this report was done under ESA contract. Responsibility for the contents resides in the Author or organisation that prepared it.

ERC32 Products

Evaluation Programme

Final Report

Status :	Final
Author :	A. Paganone _____
Authorised by :	P. Coppola _____

September 1997

Intecs Sistemi S.p.A.

TABLE OF CONTENTS

1. INTRODUCTION

This document is the Final Report for the ERC32 products evaluation activity performed by Intecs Sistemi.

The objective of the activity was to evaluate the tools aimed to support the software implementation and validation, that is the Aonix AdaWorld Development Environment, the Spacebel ERC32 Target Simulator tool, the Spacebel temporal analysis tools, namely the Schedulability Analyser and the Scheduler Simulator.

The evaluation activity was aimed to verify the suitability of each single tool to the development of (hard) real-time Ada application in the space domain and in particular to check the inter-operability of the tools over the software life cycle.

The reader is expected to have a basic knowledge of the scheduling theories, Ada and HOOD.

2. REFERENCED DOCUMENTS

[ADW-ig] Thomson Software Products, "AdaWorld Development Environment for SPARC based Workstations under Solaris 2 to ERC32 Targets, Installation Guide", MCD-ALS-P2-DOC-002.

[HRT-ig] Spacebel Informatique, "Hard Real Time Tools V2.2 Delivery".

[HRT-um] Spacebel Informatique, "32-Bit Microprocessor and Computer System Development - Schedulability Analyser and Scheduler Simulator User's Manual", 32B-SBI-SUM-0189-001/2.

[TS-ig] Spacebel Informatique, "Target Simulator V2.2 Delivery".

[TS-um] Spacebel Informatique, "32-Bit Microprocessor and Computer System Development - Target Simulator User's Manual", 32B-SBI-SUM-0189-003.

[ABRW91] N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings, "Hard Real-Time Scheduling : the Deadline-Monotonic Approach", IEEE Workshop on Real-Time Operating Systems, 1991.

[HRM92] HOOD Technical Group: "HOOD Reference Manual" - Release 3.1.1, Feb. 1992

[BWY93] A. Burns, A. Wellings, University of York, "HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems", version 2.0 Reference Manual, September 1993.

[ATB93] N. Audsley, K. Tindell, A. Burns, University of York, "The End of The Line for Static Cyclic Scheduling ?", Proceedings of the 5th Euromicro Workshop on Real-time Systems, June 1993.

[HRN95a] INTECS Sistemi S.p.A., "HRT-HoodNICE specification", July 1995.

[HRN95b] INTECS Sistemi S.p.A., "HRT-HoodNICE code extractor", June 1995.

[HRN95c] INTECS Sistemi S.p.A., "HRT-HoodNICE to HRT Tools Interface Specification", November 1995.

3. ITEMS UNDER EVALUATION

The following tools have been used along the evaluation activity:

Schedulability Analyser allowing the static verification, before execution time, of the timing constraints applying to real-time applications.

Scheduler Simulator providing the means to visualise the execution of a thread set in terms of scheduling events that occur during the modeled application's execution lifetime.

Target Simulator providing a simulator of a computer that is build around the ERC32 bit core. It allows the simulation of IU, FPU, MEC, ATAC, UART activity, watchdog, timers, interrupts and DMA transfers.

AdaWorld including the cross Ada compiler and binder. Its main innovative feature is to provide the execution time profile of the application in its worst case of execution.

AdaProbe providing symbolic debugging capabilities. By means of an appropriate configuration of the Monitor it allows to run the executable of the application on the Target Simulator.

Microtec linker available through a telnet/ftp connection to ESTEC.

All tools but the linker were installed at Intecs premises on a SPARC workstation running Solaris 2 [ADW-ig][HRT-ig][TS-ig]. The temporary evaluation licenses have been extended to the end of July 1997. All tools use FLEXLM as license manager; this allows to concentrate in a single license file all data and have the access controlled by a single instance of the manager.

Tools installation has been complex in some cases but never critical; the tools providers on-line help was quite fast and effective. Actually the most annoying problem was due to the physical support used to deliver the tools. DAT cartridges do not seem to be very reliable when they are used to move data on a different drive even if it is nominally compatible with the one used to write.

4. EVALUATION ACTIVITY STRUCTURE

The activity is composed by two steps. The first is aimed to evaluate from a very general point of view the features and the user interface of the ERC32 products. A complete check of all functionalities as declared in the associated documentation is clearly out of the scope of this evaluation; the result of the first phase can only be just the initial feeling of a new user of the toolset concerning the level of friendliness of the interface and the suitability of the major features to the development of real-time application in the space domain.

In the second phase we mainly addressed the quantitative check of a very specific point which we think to be of vital importance to make effective the benefits of the innovative features. It is the reliability of the execution time calculation performed by the compilation system, that should be quite accurate in order to ensure that the schedulability analysis gives significant results.

4.1. QUALITATIVE EVALUATION

Two subsets of the ERC32 products are analysed in the following sections; the first groups the tools (Schedulability Analyser and Scheduler Simulator) aimed to validate the dynamic behavior of the application with reference to the timing requirements.

The second group is composed by the software development and test environment, that is the Ada factory (compiler, binder, linker and debugger) plus the Target Simulator.

4.1.1. Temporal properties analysis tools

The verification of the application software against its real-time requirements is performed by means of the **Schedulability Analyser** and the **Scheduler Simulator**. Both of them work on a model of the application and of the target provided by the users in a number of files.

The schedulability analysis is executed assuming that the application dynamic behavior is conforming to a **computational model** defined by the Deadline Monotonic Scheduling Theory [ABRW91] (optionally with arbitrary deadlines) and the Immediate Priority Ceiling Inheritance blocking protocol (or optionally interrupt inhibition).

Inputs to the analysis are [HRT-um]:

the **User Configuration File** (UCF) containing a description of the main HRT attributes of the threads (CYCLIC or SPORADIC) implementing the concurrent activities of the application. It is written and maintained by the User all along the s/w life cycle.

the **Execution Skeleton File** (ESF) containing the execution profile of the threads. It defines the threads behavior in terms of calls to PROTECTED entities, sequential blocks and bounded loops in the WORST CASE OF EXECUTION. It is either written by the User or automatically generated by the compiler/binder.

the **Run Time Support Characteristics File** (RTS) containing the figures of the platform (that is target PLUS Ada kernel) relevant to scheduling such as context switch time, delay queue management time and others that globally define the execution overhead due to the priority preemptive scheduling. It is provided by the OS or Ada kernel producer.

It is worth to note that the UCF and the ESF both contain pieces of information related to the same set of threads. Conceptually the UCF contains information that are related to the application architecture, that is the type of threads (cyclic or sporadic), their deadline and period or minimum inter-arrival time. The ESF contains mainly information related to the detail design and implementation, but note that the interactions with the PROTECTED entities are an architectural matter.

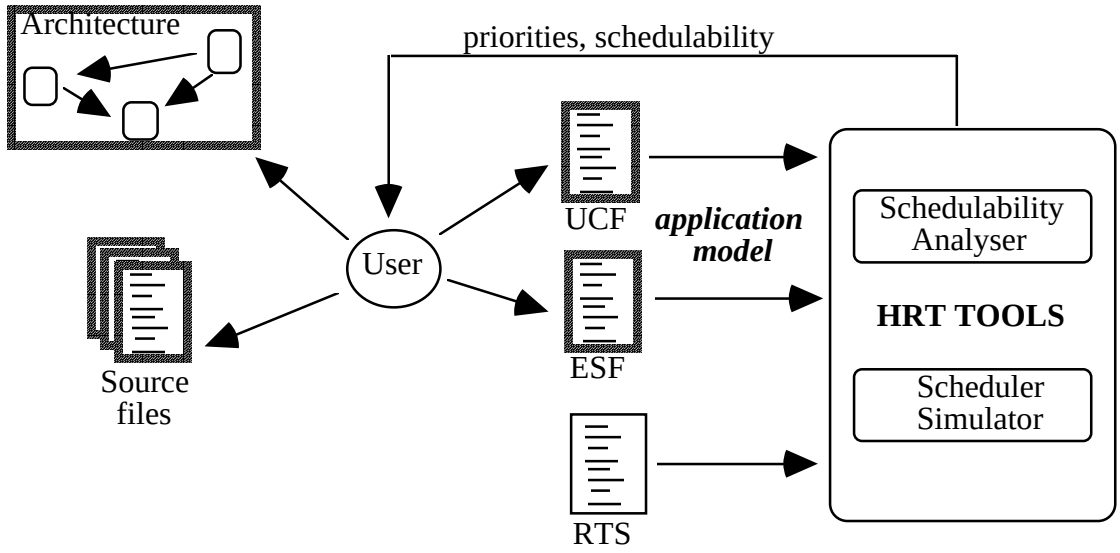
The main HRT attributes and the dynamic behavior description are kept separate, apparently in order to support the integration with the compiler/binder which is able to produce the ESF file by analysing the source code.

On the other hand when the temporal analysis tools are used without the compiler support[1] this cause a consistency problem, in that the user is expected to keep consistent two subsets of data which are logically grouped and might be specified in a single file.

In the figure the items which the User is expected to keep consistent[2] are evidenced:

[1] typically during the detailed design when no source is available, but also during the first coding phases.

[2] here it is assumed that no support tool is used for the Architecture Design and then no automatic source code extraction from the Architecture is performed.



In the following our impressions and, in some cases, suggestions:

- the temporal analysis tools have an impressive user interface. Its use is quite intuitive and comfortable.

- the diagnostics of the Schedulability Analyser when the UCF/ESF contains syntax or semantic errors could be improved. In particular it could be possible to activate the text editor on them and to embed in the files the error/warning messages as comments after the concerned row.

- the Schedulability Analyser calculates all significant figures related to scheduling. In particular the sensitivity margin is useful for the analysis of the impacts of changes to the design. It also supports arbitrary deadlines (that is, longer than the period). The main output table would be of easier comprehension referring the threads by name instead of by numbers.

- the biggest problem with the Schedulability Analyser is that its analysis algorithm does not take into account the **offset** defined for the threads. The critical instant is assumed to happen when ALL threads are released at the same time, that is the OFFSET attribute is ignored during the analysis (but it is taken into account during the simulation). The problem is that very often the applications are designed in such a way that the critical instant never happens: the activation of a thread is caused by a consequence of the actions of another thread[3] OR a separation in the time has been explicitly designed and coded in order to solve concurrent accesses to unprotected resources. In such cases the analysis performed by the Scheduler Simulator is correct

[3] as an example consider a thread which starts an action like a DMA transfer and terminates: after a while (a bounded time shorter than the thread period) an interrupt is raised to signal the end of the transfer. The (sporadic) thread in charge to manage the interrupt is never actually released at the same time as the thread which starts the transition.

but pessimistic. Then it is likely to happen that many applications that are able to work properly are declared un-schedulable because of a too pessimistic analysis. It would be important to support a schedulability check taking into account the offsets [ATB93].

- ❒ the Scheduler Simulator has been particularly appreciated for the number of simulation parameters and for the different types of reports available. The Gantt diagrams in particular are a clear representation of the simulation run easy to understand, also because of a very actractive graphics.

- ❒ the computational model is fully compatible with the HRT-HOOD method [BWY93]. From an HRT-HOOD architecture it is possible to define a model of the application simply by deriving one or two entities for each object[4]. The HRT properties of the threads match one by one the corresponding properties defined by the method for the objects.

- ❒ the tools make possible for the HRT system designer to hypothesize changes to the system and to evaluate their impacts, quickly recognising violations of the timing requirements, saving development and testing efforts (and costs).

4.1.2. Code development and test tools

AdaWorld is a quite well known Ada'83 development environment. The version included in the ERC32 Products has interesting additional features directly related to the development and validation of (hard) real-time applications.

First of all the semantics of Ada tasking has been made consistent with the scheduling model mentioned above and checks are performed to ensure that the programmer does not violate it. The computational model does not allow the real-time threads to interact directly with each other but only with PROTECTED entities.

In other words any communication or synchronisation between threads must be mediated by a Protected entity that behaves either as a synchronisation or a resource protection mechanism. A consistent implementation of the PROTECTED concept is the Ada'95 Protected Record.

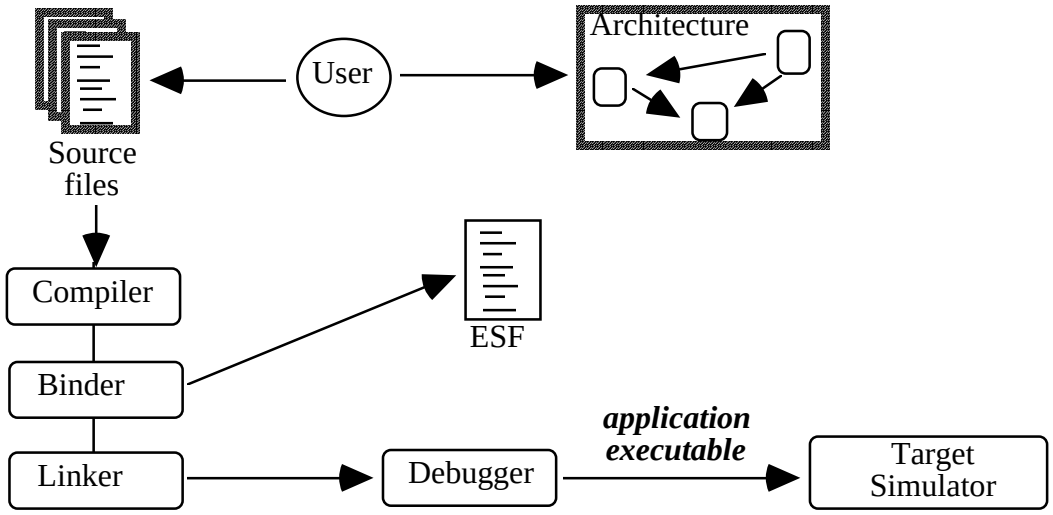
In the AdaWorld practice a **pragma Passive** is available to specify that an Ada task must behave like a Protected. All tasks which do not have the pragma Protected are considered to implement a thread. The pragma allows also an optimised implementation of the task.

The rule is that the threads can rendez-vous with protected only, while protected can only rendez-vous with each other. Then the target of a rendez-vous can be only a protected; as a consequence only the protected tasks can export entries. The only exception is a thread which is a server of an interrupt, allowed to declare a single entry which is linked to the physical signal.

---

[4] cyclic objects are represented by a cyclic thread, plus a protected resource if the object has a provided interface, sporadic objects are represented by a sporadic thread plus a protected synchronisation item.

The subset of all tasks that conform to the mentioned additional rule has the property to constitute a **predictable** concurrent application, suitable to schedulability analysis. Given that they will have a higher range of priorities than all other tasks this ensures that the schedulability analysis is a validation of the fulfillment of timing requirements.



The **compiler** detects any violation to this rule. The violation is a simple warning because the rule is applicable only to the real-time components of the application, while the other (background, lower priority) tasks are only limited by the standard Ada tasking rules. Nevertheless a violation inhibits the generation at compile time of the real-time profile of the concerned piece of code.

The second innovative feature is the ability of the code generator to 1) make a calculation of the execution time associated to the sequential pieces of code; 2) individuate the calls to protected entities made by the threads; 3) take into account iterative (bounded) statements and finally 4) to define the worst case execution path of the threads and of the protected entities. These data constitute the real-time profile of the corresponding Ada code.

By collecting the information generated at compile time the **binder** is then in a position to generate automatically the Execution Skeleton File described in the previous section. This feature is of fundamental importance in the validation of a (hard) real-time system.

Actually the preparation of a model of the application to be provided to the schedulability analysis has always been a critical point in the s/w life cycle, involving time measurements on the target, analysis of disassembled code, evaluation of the worst case path in the code and other complicated and error-prone activities.

The implementation of the mentioned features caused some **limitations** to be imposed to the code belonging to the predictable section of the application.

Most of them are in line with the chosen computational model and are mandatory if one wants to produce predictable code. For instance, only the **for .. loop** iterative statement can be used in a real-time thread and only when its parameters allow to bound the maximum number of iterations. Actually it is foreseen a specific pragma allowing the user, on its responsibility, to declare the

maximum number of iterations. This pragma (unfortunately not yet implemented) allows the use of any iterative statement.

A limitation that we do not feel to be actually needed is the lack of implementation of the **rename** statement when the entity to be renamed is an entry of a protected task. Such a construct is not allowed and it is detected as an error by the compiler. We realise that the additional rule concerning the rendez-vous is not trivial to check because any kind of indirect call (through procedures exported by other program elements for instance) must be detected, but the rename statement does not seem to be involved in this logic.

On the other hand the lack if this statement gives serious problems when the application is designed by using HOOD [HRM92] (or, more likely, HRT-HOOD). The rules to extract code from the ODS of the objects widely use the **rename**, firstly to link the operations exported by a terminal object with the entries of the Ada task implementing the dynamic behavior of the object and more in general to map the "**implemented by**" relationship between operations exported by a non-terminal objects and its child object(s) operations that actually execute the services.

This means that either the code extraction rules should be modified with degradation of the performance[5] or the architecture should be limited to be a flat architecture where all objects are terminal an no hierarchical decomposition is made.

Taking into account the wide use of (HRT-)HOOD in the space domain it is our opinion that the concerned limitation should be removed.

A further unclear point is the implementation of different **operational modes** when the ESF is produced by the compilation system. A "traditional" way to implement this feature is to code Ada tasks that at each activation[6] set their period and priority depending on the current operational mode. Furthermore they contain as the very first sequential statement a **case** on the operational mode which activates a specific subprogram corresponding to the current operational mode. Note that being different from mode to mode the HRT properties of the activities and their code, then a separate version of both the UCF and the ESF is to be maintained for each operational mode. The system must be analysed for schedulability in each mode.

On the other hand the compilation system is able to extract only one ESF from a set of source files; in particular it would generate an ESF containing the worst case of execution of the task, that is ONLY the operational mode whose associated code takes longer. In order to have one ESF for each operational mode a different set of source files should be maintained for each mode. But in this case each task can implement the behavior of the concerned activity in **one** operational mode **only** : in order to have the whole application functionality all versions of the task should be linked

---

[5] non-terminal objects are currently packages without a body that simply rename all the object's provided interface in terms of the provided interface of the child objects. To face the problem the non-terminal operations should be true subprograms that actually call child operation(s) and the package should have a body.

[6] we refer for simplicity to tasks implementing periodic activities.

in the final executable and activated when appropriate. This is, of course, impossible because of the duplication of task and entries names.

There is nothing in particular to say about the **linker** except that it was not included in the evaluation package but was available through a remote access to ESTEC computers. Such a situation caused a bottleneck in the software development process, in particular during the tuning of the code needed for the quantitative evaluation.

The need to transfer by "ftp" the output of the binder to ESTEC, run the linker through a "telnet" connection and return the executable file to our premises was a really time consuming activity. As a consequence we put a limitation on the number of versions of the application dedicated to the quantitative evaluation.

AdaProbe is a powerful and comfortable environment for symbolic debugging of Ada applications. We particularly appreciated the user interface allowing to manage quite easily a debugging session of a concurrent application and the powerful commands available for low-level target management and monitoring. In particular it is possible to set up a trace point in the code and log the CPU cycle counter when the elaboration hits the trace point. This has been particularly important during the quantitative evaluation.

The symbolic debugger was connected to the **Target Simulator** tool by means of a very general interface based on the Solaris TCP/IP service implementation [TS-um]. The Target Simulator can also run as a stand-alone tool with a greater visibility of the simulated target behavior and a deeper control of the execution environment.

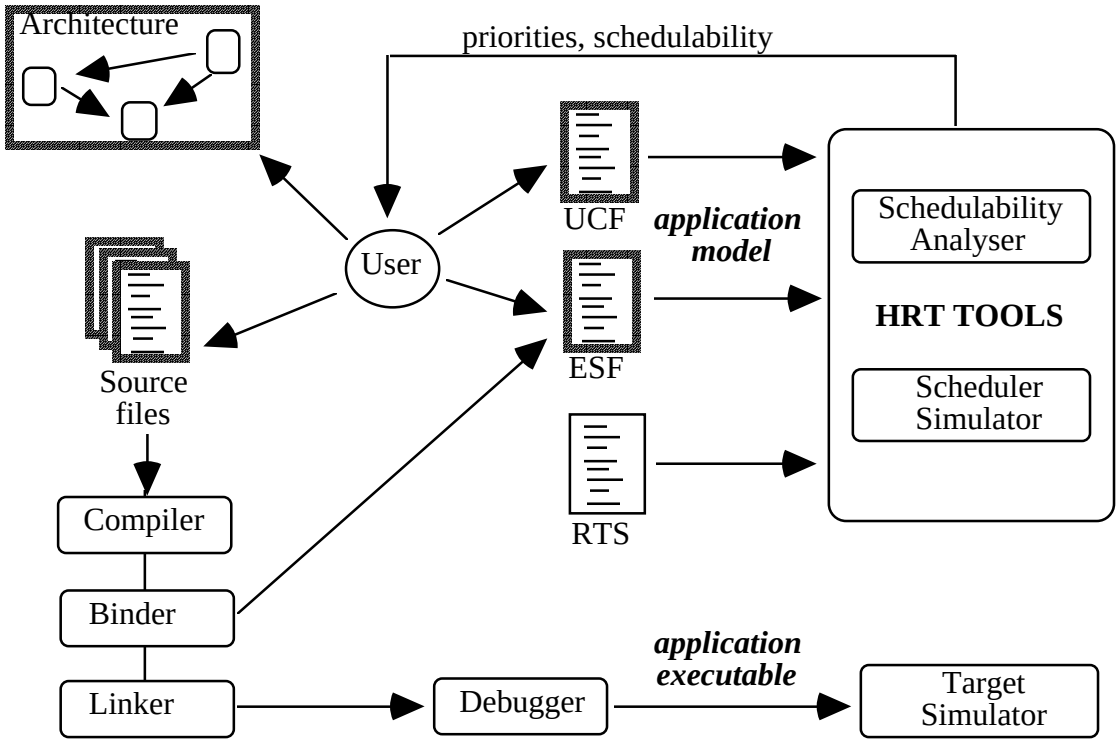
Some low level differences concerning the data/control flows to and from the Target Simulator in the two configurations caused problems during the tools installation. They were anyway quickly solved also thanks to the tools providers on-line support.

The Target Simulator appears to be a widely configurable tool, allowing the user to set-up an appropriate simulation of the processor and the surrounding hardware for a large set of actual target configurations. The full evaluation of all the features was anyway out of the scope of our evaluation activity.

4.1.3. Tools inter-operability

In this section we analyse the development environment when the whole set of ERC32 tools is used with a particular attention to the problem of keeping consistent the various involved data sets. This analysis is also put into relationship with the phases of the software life cycle.

- the UCF must be kept consistent with the Architecture all along the life cycle. We are assuming that in the design of a (hard) real-time application a design method is used that has the concept of HRT related entities like periodic/sporadic activities and protected elements. Then a mapping can be defined between architectural items and elements in the UCF. The user is expected to grant the consistency in order to be sure that the application model is representative of the actual behavior of the system.



The ESF file management is different depending on the development phase:

- during the **Architecture Design** there is no source code available and the user is expected to write "by hand" the ESF. Note that this activity is needed in order to be able to perform very early in the life cycle a preliminary schedulability analysis. The application model is quite rough at this point: the interactions between threads and protected entities are fully defined in the architecture but the architecture itself is in evolution, due also to the impact of the results of the schedulability analysis. As a consequence a number of (consistent) modifications to both the Architecture and the ESF must be foreseen. On the other hand no execution time for the sequential pieces of code is available, then the times in the ESF are estimated time or figures derived from previous experiences. In the practice at the end of this phase are available constraints to be imposed to the following phases concerning the execution times. This is anyway important because if these requirements are fulfilled by the implementation then the schedulability is granted.

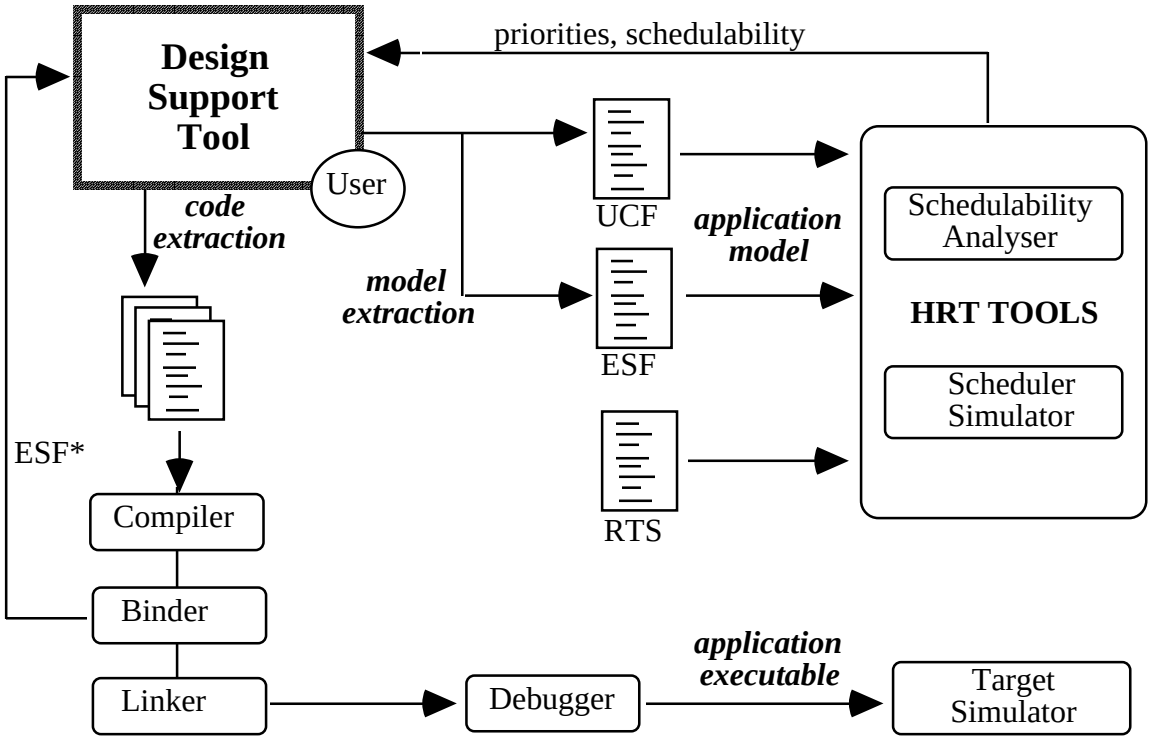
- during the **Test** phase the ESF is completely produced by the binder an then it is easy to check very often the schedulability of the system after modifications due to the debugging.

- the **Detailed Design** and **Coding** phases are the most critical from this point of view. Firstly they cannot be kept separated: the optimisation of the development effort and technical matters very often impose to code some parts of the system while others are not yet fully defined. As a consequence at a given development time some sections of the ESF can be automatically (and precisely) produced by the compilation system

while others are still under the user control. AdaWorld defines a number of additional features (pragmas) that allow to obtain a "template" of an activity not yet fully defined which has the same effect on the ESF file (with estimated times). Nevertheless it must be taken into account that in this phase a large number of changes is needed to face the first results of the application of the Test Plan and the new inputs from the schedulability analysis.

Finally, note that one complete set of all mentioned data must be maintained for each operational mode foreseen for the system (see previous section): this make the consistency almost unmanageable by hand.

To summarise it is our feeling that the ERC32 products set constitutes a good improvement of a "traditional" development environment but leaves uncovered the important problem of the consistency of a large amount of data. What is desirable is then a further tool(set) that supports the user in the design and development phases keeping a consistent representation of all the properties of the concurrent activities. In our mind such a tool can only be the **Design Support Tool**. The corresponding enhancement of the development environment is shown:



In this situation all data files (source code and model) are **extracted** form the Design Tool database and all results of the other tools are post-processed in order to be incorporated in the representation of the system. Source files and model related files are no more under the control of the User (who should not update them by hand).

The user is only expected to provide source files and model files to the temporal analysis toolset and to the code production toolset and then to activate the Design toolset features to update the

architecture on the basis of the results. Note also that a stronger **integration** between the components of the development environment might be designed.

4.2. QUANTITATIVE EVALUATION

With reference to the previous figures it is clear that a critical point in the whole design-implementation-validation process is the reliability of the calculated execution times inserted by the compilation system in the ESF file(s).

There is no check point aimed to ensure that the **application model** is actually representative of the **application executable**; the User can only rely on the correctness of the compiler-binder generated execution skeleton. In lack of a good correlation between calculated and actual times the risk is that the schedulability analysis validates a system which is different in the dynamic behavior from the actual application.

It is our opinion that it is worth to start an activity mainly aimed to:

1) check that the calculated execution times are confirmed when the application code is run on the **Target Simulator**. The Target Simulator tool provides the needed means to measure absolute values and intervals of (simulated) time. This kind of activity has been started in the second phase of our evaluation.

2) check that the calculated execution times are confirmed when the application code is run on the **actual ERC32 target**. This is also a validation step of the Target Simulator itself. This kind of activity has not been performed during our evaluation because no appropriate hardware target monitor facility was available. We think that it would be important to carry out such an evaluation in the future.

To evaluate the correlation between the calculated execution times and the corresponding (simulated) times during a run of the application we designed a little but not trivial application including the major components of a typical on-board system. It is described in §4.2.1.

The application has been designed by the HRT-HoodNICE toolset [HRN95a] based on the HRT-HOOD 2.0 method. It includes a code extractor tool tailored on the AdaWorld compiler specific features [HRN95b] and an interface to external tools able to generate a model of the application which includes (in a different format) the same information as the UCF and (partially) the ESF [HRN95c].

The toolset is able to manage a number of operational modes but because of the limitations mentioned in §4.1.2 the application has one mode only.

The UCF has been directly derived from the HRT properties of the threads by editing the automatic output of the toolset described above. It is reported in §4.2.2.

The sequential code of the threads and of the exported operation has been inserted in the toolset database and the final source code extracted. The code sections responsible for the dynamic

behavior of the threads and their interactions are **automatically** generated. The application code is shown in Appendix A.

The sequential code is only representative of an actual on-board application; it actually contains the interactions between the threads as specified in the architecture and a minimal elaboration, plus pieces of code aimed to consume CPU time.

The ESF has been produced by the AdaWorld binder. The result is a very **accurate** model of the Ada code, taking into account all details of the implementation of the HRT-HOOD objects. The time-consuming code has been then tuned in order to obtain a schedulable system with a limited margin. The final ESF is shown in §4.2.3.

Finally the executable application was run on the Target Simulator. The overall behavior was conforming to the expected one. Note that because of the minimal implementation the worst case of execution was the same as the nominal one for many threads. A set of incoming Telecommands was simulated in order to exercise all sections of the code.

After a familiarisation with the application we started the execution time measurement activity. AdaProbe allows to access low level features of the Target Simulator, in particular we used the timed trace service. It allows to log the counter of the **simulated CPU cycles** when the elaboration hits the trace points. By difference of two counter values it is possible to have a measurement of the CPU clock cycles spent in the execution of a piece of code.

Note that:

- ❒ the scheduling of the application cause no preemption between the high priority threads. The IPCI protocol ensure that the blocking time due to shared protected entities is experienced at the very beginning of the activation of each thread. The highest priority threads have periods multiple of the same time value (100 milliseconds) and then never interrupt each other. We concentrated the (simulated) time measurement activity on those threads;

- ❒ the Ada RTS does NOT use a cyclic real-time clock manager. It manages the delay queue by setting a watch-dog timer with the closest expiration time. Then no kernel thread interrupts the application threads. All delay queue management is performed when the threads execute the delay_until statement and the overhead is taken into account. The management of the watch-dog trigger is taken into account in the thread release overhead;

then it is ensured that no interference happens and the measured number of cycles is actually spent in the execution of the concerned code.

The AdaWorld binder provides in the ESF 3 values related to each piece of sequential code; the first is expressed in CPU cycles and is the number of cycles needed to execute the statements. The other values are the number of read and write accesses to memory. They allow to take into account of the wait-states needed on the specific target to access the memory. Then the total duration of the code execution **in CPU cycles** is the result of:

CPU_cycles + (1 + WS_Read) * Nof_Read + (1 + WS_Write) * Nof_Write

where WS_Read and WS_Write are the number of wait-states spent during accesses to memory. Then a row in the ESF file related to sequential code such as

WCET 1000, 200, 100

when WS_Read is 1 and WS_Write is 2 means that the corresponding code is expected to be executed in 1.700 CPU cycles. When the code is executed on the Target Simulator the same number is expected to be the difference between the two values of the CPU cycle counter.

The information about the number of wait-states is to be provided (in a consistent way) both to the Schedulability Analyser and the Target Simulator. The first finds the information in the UCF and uses it to calculate the actual execution time from the figures found in the ESF. The second finds the information in its configuration files (set up of simulated target) and uses it to increment the CPU cycles counter of the appropriate value in front of any memory access.

The ESF in §4.2.2 has been annotated with the CPU cycles resulting from the run of the application on the Target Simulator. They are reported in bold on the right side and refer to the WCET statement on the same row.

In our case the UCF declares no wait-state, both in read and write accesses. This should also be the default for the Target Simulator when the standard configuration provided into the evaluation package is used. As a consequence the measured number of CPU cycles should be simply the sum of the 3 values associated to the WCET keyword.

Actually it happens only when the number of memory accesses is very low, otherwise the measured values is **higher** than expected. In particular it is possible to see that in most cases the measured values would be consistent with the ESF figures if the number of wait-states were greater than zero (about 3 - 4). The inconsistency might be due to a number of causes:

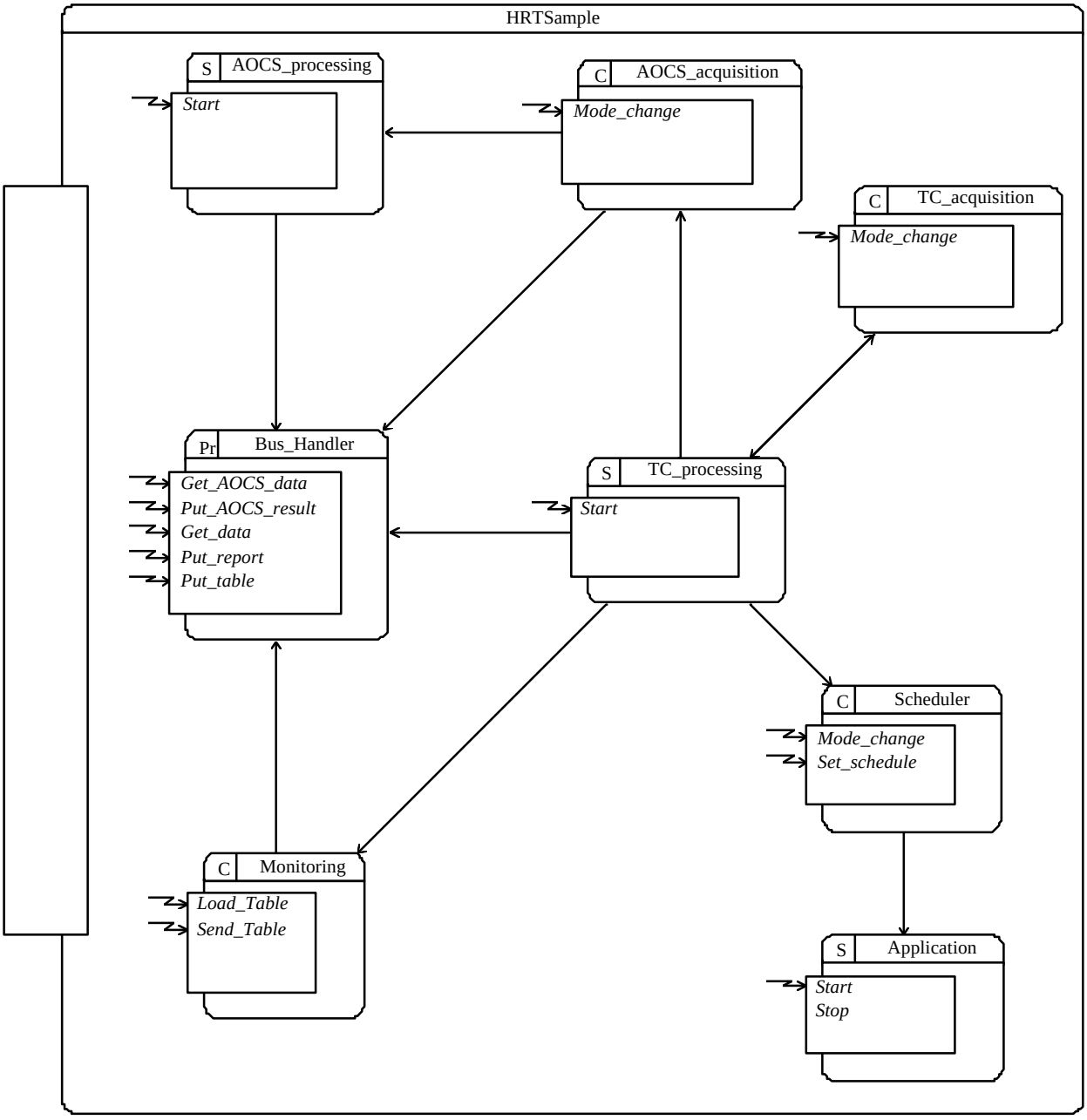
- ❒ our misunderstanding concerning the Target Simulator initialisation. The wait-states declared in the simulation environment might be erroneously different from zero;
- ❒ our fault in the measurement of the simulated CPU cycles during the simulation: there might be some overhead that we did not take into account;
- ❒ a fault in our comparison method: it is possible that the figures from simulation cannot be directly compared with figures in the ESF but a scaling factor is to be applied for reasons that we did not realise;
- ❒ a wrong processing of the initial configuration by the Target Simulator or a systematic error in the use of the wait-state values;
- ❒ some systematic error in the calculation of the number of memory accesses by the compiler.

We investigated in particular the Target Simulator initialisation and made a number of additional measurements (that confirmed the trend). We also repeated the measurements using the Target Simulator as a stand-alone tool to avoid differences due to the debugger, but we obtained the same values. Finally we had to stop the activity due to licenses expiration.

Some our error has the highest probability but it would be important to detect it in order to warn future users of the ERC32 toolset. On the other hand if some systematic error exist in the tools, then it is very important to fix it before the industrialisation phase. An activity of time measurement on the actual target would be the most appropriate.

4.2.1. The Test Application

This simple application includes threads representative of the AOCS data acquisition and processing, Telecommands execution, Monitoring activity and exchanges with other subsystems through a communication Bus.



The Monitoring activity is performed every second. It acquires data from the Bus depending on a configurable table (max 5 values) and checks them against the expected range. A diagnostic message is produced collecting all out-of-range reports, if any, within 30 m.sec.

20 application processes (FLAP) have to be scheduled within every period of 15 seconds. The overall processing time of all applications is expected to take not longer than 2 seconds. The scheduling is driven by a configurable table.

Incoming Telecommands are made available at a frequency of 1/2 Hertz during the following 20 m.sec. Telecommands are supported to : 1) change the monitoring table; 2) read the current

monitoring table and send it on the bus; 3) perform a mode change; 4) change the FLAP scheduling table.

AOCS processing is performed at 100 m.sec rate. The results must be sent on the Bus within 50 m.sec. In case of out-of range data from sensors a diagnostic message must be delivered on the Bus within 20 m.sec. and the calculation performed using the value valid during previous cycle.

The HRT-HOOD diagram of the test application is shown. AdaWorld limitations imposed to avoid any break-down and to design terminal objects only (see §4.1.2).

4.2.2. The User Configuration File

```
PREFERENCES
       DMS
       BLOCKING PROTOCOL IPCI
END

TARGET CHARACTERISTICS
       NO ATAC
       PRIORITY LOW  1
       PRIORITY HIGH 63
       CPU CLOCK FREQUENCY 10 MHz
       WAIT STATES READ 0 WRITE 0
END

THREAD DEFINITION

THREAD AOCS_acquisition.THREAD
       CRITICALITY           Hard
       PERIOD             1000000
       DEADLINE            200000
       OFFSET                   0
END AOCS_acquisition.THREAD

THREAD Scheduler.THREAD
       CRITICALITY           Hard
       PERIOD             7500000
       DEADLINE           7500000
       OFFSET                   0
END Scheduler.THREAD

THREAD Monitoring.THREAD
       CRITICALITY           Hard
       PERIOD            10000000
       DEADLINE            300000
       OFFSET                   0
END Monitoring.THREAD

THREAD TC_acquisition.THREAD
       CRITICALITY           Hard
       PERIOD             5000000
       DEADLINE            200000
       OFFSET                   0
END TC_acquisition.THREAD

THREAD TC_processing.THREAD
       CRITICALITY           Hard
       MINIMUM            5000000
       DEADLINE           4000000
END TC_processing.THREAD
```

```
THREAD Application.THREAD
       CRITICALITY           soft
       MINIMUM            7500000
       DEADLINE           7500000
END Application.THREAD

THREAD AOCS_processing.THREAD
       CRITICALITY           Hard
       MINIMUM            1000000
       DEADLINE            500000
END AOCS_processing.THREAD

END

WCET DATA
-- Text for WCET
END
```

For completeness it is also reported the Run Time Characteristics File supplied by the Ada RTS manufacturer. It contains a description of the run-time metrics parts of the computational model. It is NOT under the User control an should not be edited.

```
CONTEXT   SWITCH    TIME                              3,0,0
SCHEDULER SELECT    TIME                             (1,0,0)
TIMER     INTERRUPT OVERHEAD                          12,0,0
READY     AFTER     DELAY                             5,0,0
ENTER     DELAY     UNTIL    OVERHEAD     AT HEAD   (10,0,0)
ENTER     DELAY     UNTIL    OVERHEAD NOT AT HEAD    (2,0,0)
EXIT      DELAY     UNTIL    OVERHEAD                  1,0,0
ENTER     PASSIVE   TASK                              2,0,0
EXIT      PASSIVE   TASK                              1,0,0
ENTER     SOFTWARE  SPORADIC WAIT                     (2,0,0)
EXIT      SOFTWARE  SPORADIC WAIT                      1,0,0
ENTRY     QUEUE     SERVICING                         (3,0,0)
INTERRUPT HANDLING  OVERHEAD                          10,0,0
ENTER     INTERRUPT SPORADIC WAIT                     (2,0,0)
EXIT      INTERRUPT SPORADIC WAIT                      1,0,0

MAXIMUM NON PREEMPTION                                 0,0,0
```

4.2.3. The Execution Skeleton File

```
PROGRAM HRTSAMPLE

  THREAD TC_PROCESSING.THREAD
     TYPE SPORADIC
     WCET 3, 0, 0
     CALL_PO TC_PROCESSING.OBCS WAIT_START
     WCET 95616, 15065, 10066                                          230.456
     CALL_PO BUS_HANDLER.OBCS PUT_REPORT
     WCET 5, 0, 0
     PO SCHEDULER.OBCS SET_SCHEDULE  MONITORING.OBCS LOAD_TABLE  MONITORING.OBCS
SEND_TABLE  AOCS_ACQUISITION.OBCS MODE_CHANGE
TC_ACQUISITION.OBCS MODE_CHANGE  SCHEDULER.OBCS MODE_CHANGE
  END

  THREAD TC_ACQUISITION.THREAD
     TYPE CYCLIC
     WCET 9579, 1506, 1005                                              23.561
     CALL_PO TC_PROCESSING.OBCS START
     WCET 102, 9, 4                                                       157
     PO TC_ACQUISITION.OBCS MODE_CHANGE_PENDING  TC_ACQUISITION.OBCS
MODE_CHANGE_IN_PROGRESS  TC_ACQUISITION.OBCS END_ATC_EXECUTION
  END

  THREAD SCHEDULER.THREAD
     TYPE CYCLIC
     WCET 28694, 4519, 3020
     CALL_PO APPLICATION.OBCS START
     WCET 84, 7, 3                                                        135
     PO SCHEDULER.OBCS MODE_CHANGE_PENDING  SCHEDULER.OBCS MODE_CHANGE_IN_PROGRESS
SCHEDULER.OBCS END_ATC_EXECUTION
SCHEDULER.OBCS SET_SCHEDULE_PENDING  SCHEDULER.OBCS SET_SCHEDULE_IN_PROGRESS
  END

  THREAD MONITORING.THREAD
     TYPE CYCLIC
     WCET 24, 1, 2                                                         31
     LOOP 3
        WCET 7, 1, 1
        CALL_PO BUS_HANDLER.OBCS GET_DATA
        WCET 9551, 1505, 1005                                          23.542
     END
     WCET 2, 0, 0
     CALL_PO BUS_HANDLER.OBCS PUT_REPORT
     WCET 84, 7, 3                                                        134
     PO MONITORING.OBCS LOAD_TABLE_PENDING  MONITORING.OBCS LOAD_TABLE_IN_PROGRESS
MONITORING.OBCS END_ATC_EXECUTION
MONITORING.OBCS SEND_TABLE_PENDING  MONITORING.OBCS SEND_TABLE_IN_PROGRESS
BUS_HANDLER.OBCS PUT_TABLE
  END

  THREAD APPLICATION.THREAD
     TYPE SPORADIC
     WCET 3, 0, 0
```

```
     CALL_PO APPLICATION.OBCS WAIT_START
     WCET 955628, 150601, 100604
  END

  THREAD AOCS_PROCESSING.THREAD
     TYPE SPORADIC
     WCET 3, 0, 0
     CALL_PO AOCS_PROCESSING.OBCS WAIT_START
     WCET 143921, 22663, 15128
     CALL_PO BUS_HANDLER.OBCS PUT_AOCS_RESULT
     WCET 4, 0, 0
  END

  THREAD AOCS_ACQUISITION.THREAD
     TYPE CYCLIC
     WCET 18, 1, 0                                                    19
     CALL_PO BUS_HANDLER.OBCS GET_AOCS_DATA
     WCET 76463, 12049, 8051                                      188.336
     CALL_PO AOCS_PROCESSING.OBCS START
     WCET 84, 7, 3                                                   134
     PO AOCS_ACQUISITION.OBCS MODE_CHANGE_PENDING  AOCS_ACQUISITION.OBCS
MODE_CHANGE_IN_PROGRESS  AOCS_ACQUISITION.OBCS
END_ATC_EXECUTION
  END

  PROTECTED TC_PROCESSING.OBCS
     TYPE SYNCHRO
     ENTRY START
        WCET 75, 10, 10                                              148
     BARRIER WCET 6, 1, 0
     ENTRY WAIT_START
        WCET 43, 6, 6
  END

  PROTECTED TC_ACQUISITION.OBCS
     TYPE RESOURCE
     ENTRY END_ATC_EXECUTION
        WCET 33, 2, 1
     ENTRY MODE_CHANGE_IN_PROGRESS
        WCET 20, 0, 1
     ENTRY MODE_CHANGE_PENDING
        WCET 24, 2, 1
     ENTRY MODE_CHANGE
        WCET 37, 2, 2
  END
```

```
PROTECTED SCHEDULER.OBCS
   TYPE RESOURCE
   ENTRY END_ATC_EXECUTION
      WCET 33, 2, 1
   ENTRY SET_SCHEDULE_IN_PROGRESS
      WCET 20, 0, 1
   ENTRY SET_SCHEDULE_PENDING
      WCET 24, 2, 1
   ENTRY SET_SCHEDULE
      WCET 37, 2, 2
   ENTRY MODE_CHANGE_IN_PROGRESS
      WCET 20, 0, 1
   ENTRY MODE_CHANGE_PENDING
      WCET 24, 2, 1
   ENTRY MODE_CHANGE
      WCET 37, 2, 2
END

PROTECTED MONITORING.OBCS
   TYPE RESOURCE
   ENTRY END_ATC_EXECUTION
      WCET 33, 2, 1
   ENTRY SEND_TABLE_IN_PROGRESS
      WCET 20, 0, 1
   ENTRY SEND_TABLE_PENDING
      WCET 24, 2, 1
   ENTRY SEND_TABLE
      WCET 37, 2, 2
   ENTRY LOAD_TABLE_IN_PROGRESS
      WCET 20, 0, 1
   ENTRY LOAD_TABLE_PENDING
      WCET 24, 2, 1
   ENTRY LOAD_TABLE
      WCET 37, 2, 2
END

PROTECTED BUS_HANDLER.OBCS
   TYPE RESOURCE
   ENTRY PUT_TABLE
      WCET 9566, 1503, 1004
   ENTRY PUT_REPORT
      WCET 9566, 1503, 1004
   ENTRY GET_DATA
      WCET 9566, 1503, 1004
   ENTRY PUT_AOCS_RESULT
      WCET 28710, 4519, 3022
   ENTRY GET_AOCS_DATA
      WCET 28733, 4523, 3023
END
```

```
  PROTECTED APPLICATION.OBCS
     TYPE SYNCHRO
     ENTRY START
        WCET 60, 7, 7
     BARRIER WCET 6, 1, 0
     ENTRY WAIT_START
        WCET 38, 5, 5
  END

  PROTECTED AOCS_PROCESSING.OBCS
     TYPE SYNCHRO
     ENTRY START
        WCET 73, 9, 10                                              142
     BARRIER WCET 6, 1, 0
     ENTRY WAIT_START
        WCET 43, 6, 6                                                75
  END

  PROTECTED AOCS_ACQUISITION.OBCS
     TYPE RESOURCE
     ENTRY END_ATC_EXECUTION
        WCET 33, 2, 1
     ENTRY MODE_CHANGE_IN_PROGRESS
        WCET 20, 0, 1
     ENTRY MODE_CHANGE_PENDING
        WCET 24, 2, 1
     ENTRY MODE_CHANGE
        WCET 37, 2, 2
  END
END
```

5. CONCLUSIONS

The ERC32 products constitute an actractive and innovative development environment. They include features of vital importance for an efficient and effective support to the design, development and validation of (hard) real-time systems all along the software life cycle.

Each tool is appreciable for what concern the overall conception and user interface. Many tools provide features that are on the top of the current industrial trends.

Nevertheless it is our opinion that some specific aspects should be improved in order to experiment the full benefit of an integrated development environment. They are listed in the following, in importance order:

- **Operational modes management** : some automatic support is needed for the management of the different operational modes that are generally designed for any real-time system. Such a support should allow to use the automatic ESF generation and save the User from the need to take into account "by hand" the different instances of each thread.

- **Data consistency** : as shown a large amount of data must be kept consistent over the design, development and test phases. The management of the operational modes adds a further complexity level. The resulting check and alignment activity is complicated and error-prone when left completely on the User responsibility. The automatic support mentioned above should be extended to other aspects of the inter-operability of the ERC32 products. It is our opinion that a Design Support Tool is the most appropriate location for this kind of support.

- **Computational models** : enhanced computational models taking into account threads offsets and the corresponding schedulability check algorithms should be used. They allow to model properly a wider range of software applications.

- **Language limitations** : additional rules and limitations to the coding activity should be applied only when strictly needed.

Finally we suggest some additional action to clarify the (apparent) inconsistencies related to the sequential code execution times calculation. Such an activity should also be extended to the concurrent sections of the code in order to ensure that the ESF is fully representative of the actual application code behavior.

6. ACRONYMS

ACS	Ada Compilation System
ADD	Architectural Design Document
ASER	ASynchronous Execution Request
DDD	Detailed Design Document
DHS	Data Handling System
DMST	Deadline Monotonic Scheduling Theory
ESA	European Space Agency
ESTEC	European Space Technical Center
HOOD	Hierarchical Object Oriented Design
HRT	Hard Real Time
IPCI	Immediate Priority Ceiling Inheritance
OBCS	Object Control Structure
ODS	Object Description Skeleton
RMST	Rate Monotonic Scheduling Theory
RTS	Run Time Support
SDE	Software Development Environment

APPENDIX A - TEST APPLICATION SOURCE CODE

```
package AOCS_data is    -- PASSIVE ENVIRONMENT HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
     -------------   TYPE(S)   ------------
   type AOCS_raw_data  is array ( 1 .. 10 ) of Positive;
   type Item  is record
          Data_type : Positive range 1 .. 5;
          The_data : AOCS_raw_data;
        end record;
   subtype Index  is Positive range 1 .. 5;
   type List  is array (Index) of Item;
   subtype AOCS_result  is Integer range 0 .. 1000;
     -----------   CONSTANT(S)   ----------
   The_list : constant  List := (
          1 => ( 1, ( others => 20)),
          2 => ( 2, ( others => 30)),
          3 => ( 3, ( others => 40)),
          4 => ( 4, ( others => 50)),
          5 => ( 5, ( others => 60))
        );
end AOCS_data;

package Telecommand is  -- PASSIVE ENVIRONMENT HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
     -------------   TYPE(S)   ------------
   type Item  is record
          OP_code : Positive;
        end record;
   type Index  is range 1 .. 10;
   type List  is array ( Index ) of Item;
     -----------   CONSTANT(S)   ----------
   The_list : constant  List := (
          1 => ( Op_code => 1),
          2 => ( Op_code => 2),
          3 => ( Op_code => 3),
          4 => ( Op_code => 4),
          5 => ( Op_code => 5),
          6 => ( Op_code => 6),
          7 => ( Op_code => 7),
          8 => ( Op_code => 8),
          9 => ( Op_code => 9),
          10 => ( Op_code => 10)
        );
end Telecommand;
```

```
-- Alsys compiler version

with System; use System;
with Real_time; use Real_time;
with Interrupt_Manager; use Interrupt_Manager;
package RTA is
     -------------   TYPE(S)   ------------
   type SYSTEM_WIDE_MODE  is ( Mode_1 , Mode_2 , Mode_3 );
   type T_IMPORTANCE      is ( HARD , SOFT , BACKGROUND );
   type MODE_PRIORITY     is array ( SYSTEM_WIDE_MODE ) of PRIORITY;
   type MODE_DURATION     is array ( SYSTEM_WIDE_MODE ) of Time_Span;
   type MODE_IMPORTANCE   is array ( SYSTEM_WIDE_MODE ) of T_IMPORTANCE;
     -----------   CONSTANT(S)   ----------
   START_MODE : constant  SYSTEM_WIDE_MODE := SYSTEM_WIDE_MODE'first;
   DEFAULT_PRIORITY : constant  PRIORITY         := PRIORITY'first;
   --
   INT0  : constant INTERRUPT_ID :=  0;
   INT1  : constant INTERRUPT_ID :=  1;
   INT2  : constant INTERRUPT_ID :=  2;
   INT3  : constant INTERRUPT_ID :=  3;
   INT4  : constant INTERRUPT_ID :=  4;
   INT5  : constant INTERRUPT_ID :=  5;
   INT6  : constant INTERRUPT_ID :=  6;
   INT7  : constant INTERRUPT_ID :=  7;
   INT8  : constant INTERRUPT_ID :=  8;
   INT9  : constant INTERRUPT_ID :=  9;
   INT10 : constant INTERRUPT_ID := 10;
   INT11 : constant INTERRUPT_ID := 11;
   INT12 : constant INTERRUPT_ID := 12;
   INT13 : constant INTERRUPT_ID := 13;
   INT14 : constant INTERRUPT_ID := 14;
   INT15 : constant INTERRUPT_ID := 15;
   --
     ----------   OPERATION(S)   ----------
   function CURRENT_MODE return SYSTEM_WIDE_MODE;

   procedure SET_CURRENT_MODE( To_Value : in SYSTEM_WIDE_MODE );

   function SYSTEM_START_UP_TIME return TIME;

   procedure SET_SYSTEM_START_UP_TIME( To_Value : in TIME );

   procedure EXE_1_MSEC;
end RTA;
package body RTA is

   MILLISECOND : constant Integer := 500;
   Dummy     : Integer := 0;
   Flag   : BOOLEAN := FALSE;

   The_mode  : SYSTEM_WIDE_MODE   := START_MODE;
   Start_time  : TIME             := Clock;

     ----------   OPERATION(S)   ----------

   function CURRENT_MODE return SYSTEM_WIDE_MODE is
   begin
     return The_mode;
   end CURRENT_MODE;

   procedure SET_CURRENT_MODE( To_Value : in SYSTEM_WIDE_MODE ) is
   begin
     The_mode := To_Value;
   end SET_CURRENT_MODE;
```

```
   function SYSTEM_START_UP_TIME return TIME is
   begin
     return Start_time;
   end SYSTEM_START_UP_TIME;

   procedure SET_SYSTEM_START_UP_TIME( To_Value : in TIME ) is
   begin
     Start_time := To_Value;
   end SET_SYSTEM_START_UP_TIME;

   procedure EXE_1_MSEC is
   begin
     for Index in 1 .. MILLISECOND loop
       Flag := not Flag;
       if Flag then
         Dummy := Dummy + 1;
       else
         Dummy := Dummy - 1;
       end if;
     end loop;
   end EXE_1_MSEC;

end RTA;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package AOCS_acquisition_RTATT is
   CEILING   : constant MODE_PRIORITY := (
          others => 14);
   TRANSFORMATION   : constant INTEGER := 0;
   THREAD_PERIOD    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(100.000 / 1000));
   THREAD_OFFSET    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(0.000 / 1000));
   THREAD_DEADLINE  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(20.000 / 1000));
   THREAD_PRIORITY  : constant MODE_PRIORITY := (
          others => 12);
   THREAD_IMPORTANCE     : constant MODE_IMPORTANCE := (
          others => hard);
   INITIAL_PRIORITY :   constant PRIORITY := 12;
   INITIAL_CEILING :    constant PRIORITY := 14;
   Mode_change_WCET : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
   thread_action_WCET    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(8.000 / 1000));
end AOCS_acquisition_RTATT;

--HSATC operations are executed by the OBCS task inside the rendez-vous.
--The OBCS exports one entry for each HSATC provided operation. They are
--   entered by calling user(s).
--LSATC and ASATC operations are executed by the THREAD task.
--The OBCS exports 3 entries for each LSATC or ASATC. One is similar to
--   the HSATC entries, the others are called only by the THREAD task.
--   They have the purpose to manage in mutual exclusion the parameters
--   buffers and the ATC related status variables.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
```

```
with AOCS_acquisition_RTATT;
package AOCS_acquisition is      -- CYCLIC HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
   ---------------- OBJECT CONTROL STRUCTURE    ----------------
     --------  DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       -- Mode_change constrained by ASATC

   task OBCS is
     pragma PRIORITY(AOCS_acquisition_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Mode_change;
     -- called by user object(s)
     entry Mode_change_IN_PROGRESS;
     -- called by THREAD task only
     entry Mode_change_PENDING(IS_PENDING : out BOOLEAN);
     -- called by THREAD task only
     entry END_ATC_EXECUTION;
     -- called by THREAD task only
   end OBCS;
end AOCS_acquisition;

with Real_Time; use Real_Time;    -- Alsys time management services
   ---------------- REQUIRED OBJECT : Bus_Handler    ----------------
     ----------   OPERATION(S)   ----------
   -- Get_AOCS_data
with Bus_Handler;
   ---------------- REQUIRED OBJECT : AOCS_processing----------------
     ----------   OPERATION(S)   ----------
   -- Start
with AOCS_processing;
   ---------------- REQUIRED OBJECT : AOCS_data ----------------
     -------------   TYPE(S)   ------------
   -- Index
with AOCS_data;

package body AOCS_acquisition is      -- CYCLIC HRT-HOOD object implementation
----------------   INTERNAL DECLARATIONS   ----------------
     ----------   OPERATION(S)   ----------
   procedure thread_action;
   --------   FORWARD OPERATION DECLARATIONS   ----------
   procedure OPCS_Mode_change;

   ---------------- OBJECT CONTROL STRUCTURE    ----------------
   --|:OBCS_CODE
   --|:END_CODE
   -- declarations to support LSATC and ASATC operations
   -- operation Mode_change, unbuffered
   Mode_change_CALLED : BOOLEAN := FALSE;

   -- data for ATC management
   ATC_IN_PROGRESS : NATURAL := 0;

   task THREAD is
     pragma PRIORITY( AOCS_acquisition_RTATT.INITIAL_PRIORITY);
   end THREAD;

   task body THREAD is
     T    : Real_Time.TIME := Real_Time.CLOCK;
     ATC_PENDING        : BOOLEAN;
   begin
     if T < RTA.SYSTEM_START_UP_TIME then
       T := RTA.SYSTEM_START_UP_TIME;
```

```
   end if;
   T := T + AOCS_acquisition_RTATT.THREAD_OFFSET(RTA.START_MODE);
   FOREVER:
   loop
     Real_Time.delay_until(T);
     if ATC_IN_PROGRESS > 0 then
       OBCS.Mode_change_PENDING(ATC_PENDING);
       if ATC_PENDING then
         OBCS.Mode_change_IN_PROGRESS;
         OPCS_Mode_change;
         OBCS.END_ATC_EXECUTION;
       else
         null; -- should never be executed
       end if;
     else
       thread_action;
     end if;
     T := T + AOCS_acquisition_RTATT.THREAD_PERIOD(RTA.CURRENT_MODE);
   end loop FOREVER;
end THREAD;

task body OBCS is
begin
   loop
     select
       accept Mode_change do
         -- if not already pending prepare execution,
         -- otherwise over-write previous request.
         if not Mode_change_CALLED then
           Mode_change_CALLED := TRUE;
           ATC_IN_PROGRESS := ATC_IN_PROGRESS + 1;
         end if;
       end Mode_change;
     or
       accept Mode_change_PENDING(IS_PENDING : out BOOLEAN) do
         IS_PENDING := Mode_change_CALLED;
       end Mode_change_PENDING;
     or
       accept Mode_change_IN_PROGRESS do
         Mode_change_CALLED := FALSE;
       end Mode_change_IN_PROGRESS;
     or
       accept END_ATC_EXECUTION do
         if ATC_IN_PROGRESS > 0 then    -- ignore first call (initialisation)
           ATC_IN_PROGRESS := ATC_IN_PROGRESS - 1;
         end if;
       end END_ATC_EXECUTION;
     or
       terminate;
     end select;
   end loop;
end OBCS;

--------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

procedure OPCS_Mode_change is
--|:OPCS_CODE <Mode_change>
   begin
     EXE_1_MSEC;
   --|:END_CODE
end OPCS_Mode_change;

---------------  OPCS OF UNCONSTRAINED OPERATIONS     ----------------
```

```
   procedure thread_action is
   --|:OPCS_CODE <thread_action>
        Last_AOCS_data : AOCS_data.Index;
     begin
        Bus_Handler.OBCS.Get_AOCS_data(Last_AOCS_data);
        for I in 1 .. 8 loop
          EXE_1_MSEC;
        end loop;
        AOCS_processing.OBCS.Start(Last_AOCS_data);
     --|:END_CODE
   end thread_action;
end AOCS_acquisition;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package AOCS_processing_RTATT is
   CEILING   : constant MODE_PRIORITY := (
          others => 13);
   THREAD_MAT  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(100.000 / 1000));
   THREAD_DEADLINE  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(50.000 / 1000));
   THREAD_PRIORITY  : constant MODE_PRIORITY := (
          others => 6);
   THREAD_IMPORTANCE     : constant MODE_IMPORTANCE := (
          others => hard);
   INITIAL_PRIORITY :    constant PRIORITY := 6;
   INITIAL_CEILING :     constant PRIORITY := 13;
   Start_WCET  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(18.000 / 1000));
end AOCS_processing_RTATT;

--HSATC operations are executed by the OBCS task inside the rendez-vous.
--The OBCS exports one entry for each HSATC provided operation. They are
--   entered by calling user(s).
--LSATC and ASATC operations are executed by the THREAD task.
--The OBCS exports 3 entries for each LSATC or ASATC. One is similar to
--   the HSATC entries, the others are called only by the THREAD task.
--   They have the purpose to manage in mutual exclusion the parameters
--   buffers and the ATC related status variables.


with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
with AOCS_processing_RTATT;
   ---------------- REQUIRED OBJECT : AOCS_data ----------------
     -------------   TYPE(S)   ------------
   --AOCS_result
   --Item
     -----------   CONSTANT(S)   ----------
   --The_list
     -------------   TYPE(S)   ------------
   --List
   --Index
   --AOCS_raw_data
with AOCS_data;
package AOCS_processing is       -- SPORADIC HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
```

```
     ----   TYPES FOR CONSTRAINED OPS  ----
   type Start_PARAMETER_SET is record
     On_data:  AOCS_data.Index;
     OVERRUN:  BOOLEAN := FALSE;
     Not_used  :    Real_Time.Time_Span;
   end record;
   ---------------- OBJECT CONTROL STRUCTURE    ----------------
     --------  DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       -- Start constrained by ASER

   task OBCS is
     pragma PRIORITY(AOCS_processing_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Start(On_data : AOCS_data.Index);
     -- called by user object(s)
     entry WAIT_Start(THE_PARAMS : out Start_PARAMETER_SET);
     -- called by THREAD task only
   end OBCS;
end AOCS_processing;

   ---------------- REQUIRED OBJECT : Bus_Handler    ----------------
     ----------   OPERATION(S)   ----------
   -- Put_AOCS_result
with Bus_Handler;

package body AOCS_processing is       -- SPORADIC HRT-HOOD object implementation
   --------   FORWARD OPERATION DECLARATIONS   ----------
   procedure OPCS_Start(On_data : AOCS_data.Index ; OVERRUN : in BOOLEAN ; Not_used
: in Real_Time.Time_Span);

   ---------------- OBJECT CONTROL STRUCTURE    ----------------
   --|:OBCS_CODE
   --|:END_CODE
   -- operation Start, unbuffered
   Start_PARAMETERS : Start_PARAMETER_SET;
   Start_CALLED : BOOLEAN := FALSE;

   task THREAD is
     pragma PRIORITY( AOCS_processing_RTATT.INITIAL_PRIORITY);
   end THREAD;

   task body THREAD is
     Start_BUFFER  :    Start_PARAMETER_SET;
   begin
     FOREVER:
     loop
       OBCS.WAIT_Start(Start_BUFFER);
         OPCS_Start(Start_BUFFER.On_data,
                    Start_BUFFER.OVERRUN,
                    Start_BUFFER.Not_used);
     end loop FOREVER;
   end THREAD;

   task body OBCS is
   begin
     loop
       select
         accept Start(On_data : AOCS_data.Index) do
           -- store parameters
           Start_PARAMETERS := (On_data, FALSE, Real_Time.Time_Span_Zero);
           -- if not already pending prepare execution,
           -- otherwise over-write previous request.
           if not Start_CALLED then
```

```
            Start_CALLED := TRUE;
          else
            Start_PARAMETERS.OVERRUN := TRUE;
          end if;
        end Start;
      or
      when Start_Called =>
        accept WAIT_Start(THE_PARAMS : out Start_PARAMETER_SET) do
          THE_PARAMS := Start_PARAMETERS;
          Start_PARAMETERS.OVERRUN := FALSE;
          Start_CALLED := FALSE;
        end WAIT_Start;
      or
        terminate;
      end select;
    end loop;
  end OBCS;

  --------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

  procedure OPCS_Start(On_data : AOCS_data.Index ; OVERRUN : in BOOLEAN ; Not_used
: in Real_Time.Time_Span) is
  --|:OPCS_CODE <Start in AOCS_data>
      use AOCS_data;
      The_result : Integer := 0;
    begin
      for I in 1 .. 15 loop
        EXE_1_MSEC;
      end loop;
      for Index in AOCS_raw_data'range loop
        The_result := The_result + The_list(On_data).The_data(Index);
      end loop;
      Bus_Handler.OBCS.Put_AOCS_result(The_result);
    --|:END_CODE
  end OPCS_Start;
end AOCS_processing;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package Application_RTATT is
  CEILING   : constant MODE_PRIORITY := (
         others => 3);
  THREAD_MAT  : constant MODE_DURATION := (
         others => Real_Time.To_Time_Span(750.000 / 1000));
  THREAD_DEADLINE  : constant MODE_DURATION := (
         others => Real_Time.To_Time_Span(750.000 / 1000));
  THREAD_PRIORITY  : constant MODE_PRIORITY := (
         others => 1);
  THREAD_IMPORTANCE     : constant MODE_IMPORTANCE := (
         others => soft);
  INITIAL_PRIORITY :   constant PRIORITY := 1;
  INITIAL_CEILING :    constant PRIORITY := 3;
  Start_WCET  : constant MODE_DURATION := (
         others => Real_Time.To_Time_Span(100.000 / 1000));
  Stop_WCET : constant MODE_DURATION := (
         others => Real_Time.To_Time_Span(1.000 / 1000));
end Application_RTATT;

--HSATC operations are executed by the OBCS task inside the rendez-vous.
--The OBCS exports one entry for each HSATC provided operation. They are
```

```
--   entered by calling user(s).
--LSATC and ASATC operations are executed by the THREAD task.
--The OBCS exports 3 entries for each LSATC or ASATC. One is similar to
--   the HSATC entries, the others are called only by the THREAD task.
--   They have the purpose to manage in mutual exclusion the parameters
--   buffers and the ATC related status variables.


with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
with Application_RTATT;
package Application is      -- SPORADIC HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
     ----   TYPES FOR CONSTRAINED OPS  ----
   type Start_PARAMETER_SET is record
     OVERRUN:  BOOLEAN := FALSE;
     Not_used  :    Real_Time.Time_Span;
   end record;
   ---------------- OBJECT CONTROL STRUCTURE    ----------------
     --------  DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       --Start constrained by ASER

   task OBCS is
     pragma PRIORITY(Application_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Start;
     -- called by user object(s)
     entry WAIT_Start(THE_PARAMS : out Start_PARAMETER_SET);
     -- called by THREAD task only
   end OBCS;
     ----------   OPERATION(S)   ----------
   procedure Stop;
end Application;


package body Application is      -- SPORADIC HRT-HOOD object implementation
   --------   FORWARD OPERATION DECLARATIONS   ----------
   procedure OPCS_Start(OVERRUN : in BOOLEAN ; Not_used : in Real_Time.Time_Span);

   ---------------- OBJECT CONTROL STRUCTURE    ----------------
   --|:OBCS_CODE
   --|:END_CODE
   -- operation Start, unbuffered
   Start_PARAMETERS : Start_PARAMETER_SET;
   Start_CALLED : BOOLEAN := FALSE;

   task THREAD is
     pragma PRIORITY( Application_RTATT.INITIAL_PRIORITY);
   end THREAD;

   task body THREAD is
     Start_BUFFER  :    Start_PARAMETER_SET;
   begin
     FOREVER:
     loop
       OBCS.WAIT_Start(Start_BUFFER);
         OPCS_Start(Start_BUFFER.OVERRUN,
                    Start_BUFFER.Not_used);
     end loop FOREVER;
   end THREAD;
```

```
   task body OBCS is
   begin
      loop
        select
          accept Start do
            -- store parameters
            Start_PARAMETERS := (FALSE, Real_Time.Time_Span_Zero);
            -- if not already pending prepare execution,
            -- otherwise over-write previous request.
            if not Start_CALLED then
              Start_CALLED := TRUE;
            else
              Start_PARAMETERS.OVERRUN := TRUE;
            end if;
          end Start;
        or
        when Start_Called =>
          accept WAIT_Start(THE_PARAMS : out Start_PARAMETER_SET) do
            THE_PARAMS := Start_PARAMETERS;
            Start_PARAMETERS.OVERRUN := FALSE;
            Start_CALLED := FALSE;
          end WAIT_Start;
        or
          terminate;
        end select;
      end loop;
   end OBCS;

   --------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

   procedure OPCS_Start(OVERRUN : in BOOLEAN ; Not_used : in Real_Time.Time_Span) is
   --|:OPCS_CODE <Start>
      begin
        for I in 1 .. 100 loop
          EXE_1_MSEC;
        end loop;
      --|:END_CODE
   end OPCS_Start;

   ---------------  OPCS OF UNCONSTRAINED OPERATIONS     ----------------

   procedure Stop is
   --|:OPCS_CODE <Stop>
      begin
        EXE_1_MSEC;
      --|:END_CODE
   end Stop;
end Application;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package Bus_Handler_RTATT is
   CEILING   : constant MODE_PRIORITY := (
          others => 15);
   INITIAL_CEILING :    constant PRIORITY := 15;
   Get_AOCS_data_WCET   : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(3.000 / 1000));
   Put_AOCS_result_WCET : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(3.000 / 1000));
   Get_data_WCET    : constant MODE_DURATION := (
```

```
          others => Real_Time.To_Time_Span(1.000 / 1000));
   Put_report_WCET  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
   Put_table_WCET   : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
end Bus_Handler_RTATT;

-- The OBCS task accepts and executes the calls to PAER and PSER operations.
--   It also implements the functional constraints using guards.
-- For each PSER operation with functional constraints the user is expected
--   to define in the ODS internals either a parameterless function named
--   OPCS_PSER_FAC returning a BOOLEAN value or a BOOLEAN variable with
--   the same name. It will be used as guard.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Bus_Handler_RTATT;
   ---------------- REQUIRED OBJECT : AOCS_data ----------------
     -------------   TYPE(S)   ------------
   --Index
with AOCS_data;
package Bus_Handler is      -- PROTECTED HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
   ---------------- OBJECT CONTROL STRUCTURE    ----------------
     --------  DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       --Get_AOCS_data constrained by PSER
       --Put_AOCS_result constrained by PAER
       --Get_data constrained by PSER
       --Put_report constrained by PAER
       --Put_table constrained by PAER

   task OBCS is
     pragma PRIORITY(Bus_Handler_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Get_AOCS_data(The_index : out AOCS_data.Index);
     -- called by user object(s)
     entry Put_AOCS_result(The_result : AOCS_data.AOCS_result);
     -- called by user object(s)
     entry Get_data;
     -- called by user object(s)
     entry Put_report;
     -- called by user object(s)
     entry Put_table;
     -- called by user object(s)
   end OBCS;
end Bus_Handler;


package body Bus_Handler is      -- PROTECTED HRT-HOOD object implementation
----------------   INTERNAL DECLARATIONS   ----------------
     -------------   DATA   -------------
   AOCS_index :  AOCS_data.Index := AOCS_data.Index'first;
   --------   FORWARD OPERATION DECLARATIONS   ----------
   procedure OPCS_Get_AOCS_data(The_index : out AOCS_data.Index);
   -- standard operation to contain user code
   procedure OPCS_Put_AOCS_result(The_result : AOCS_data.AOCS_result);
   -- standard operation to contain user code
   procedure OPCS_Get_data;
   -- standard operation to contain user code
   procedure OPCS_Put_report;
   -- standard operation to contain user code
   procedure OPCS_Put_table;
```

```
-- standard operation to contain user code

---------------- OBJECT CONTROL STRUCTURE    ----------------
--|:OBCS_CODE
--|:END_CODE

task body OBCS is
begin
   loop
     select
       accept Get_AOCS_data(The_index : out AOCS_data.Index) do
         OPCS_Get_AOCS_data(The_index);
       end Get_AOCS_data;
     or
       accept Put_AOCS_result(The_result : AOCS_data.AOCS_result) do
         OPCS_Put_AOCS_result(The_result);
       end Put_AOCS_result;
     or
       accept Get_data do
         OPCS_Get_data;
       end Get_data;
     or
       accept Put_report do
         OPCS_Put_report;
       end Put_report;
     or
       accept Put_table do
         OPCS_Put_table;
       end Put_table;
     or
       terminate;
     end select;
   end loop;
end OBCS;

--------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

procedure OPCS_Get_AOCS_data(The_index : out AOCS_data.Index) is
--|:OPCS_CODE <Get_AOCS_data out AOCS_data>
     use AOCS_data;
   begin
     for I in 1 .. 3 loop
       EXE_1_MSEC;
     end loop;
     The_index := AOCS_index;
     if AOCS_index < AOCS_data.Index'last then
       AOCS_index := AOCS_index + 1;
     else
       AOCS_index := AOCS_data.Index'first;
     end if;
   --|:END_CODE
end OPCS_Get_AOCS_data;

procedure OPCS_Put_AOCS_result(The_result : AOCS_data.AOCS_result) is
--|:OPCS_CODE <Put_AOCS_result in AOCS_data>
   begin
     for I in 1 .. 3 loop
       EXE_1_MSEC;
     end loop;
   --|:END_CODE
end OPCS_Put_AOCS_result;

procedure OPCS_Get_data is
--|:OPCS_CODE <Get_data>
```

```
     begin
        EXE_1_MSEC;
     --|:END_CODE
   end OPCS_Get_data;

   procedure OPCS_Put_report is
   --|:OPCS_CODE <Put_report>
     begin
        EXE_1_MSEC;
     --|:END_CODE
   end OPCS_Put_report;

   procedure OPCS_Put_table is
   --|:OPCS_CODE <Put_table>
     begin
        EXE_1_MSEC;
     --|:END_CODE
   end OPCS_Put_table;
end Bus_Handler;
with TC_acquisition;
procedure HRTSample is
begin
   null;
end HRTSample;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package Monitoring_RTATT is
   CEILING   : constant MODE_PRIORITY := (
          others => 8);
   TRANSFORMATION   : constant INTEGER := 0;
   THREAD_PERIOD    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1000.000 / 1000));
   THREAD_OFFSET    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(0.000 / 1000));
   THREAD_DEADLINE  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(30.000 / 1000));
   THREAD_PRIORITY  : constant MODE_PRIORITY := (
          others => 7);
   THREAD_IMPORTANCE     : constant MODE_IMPORTANCE := (
          others => hard);
   INITIAL_PRIORITY :    constant PRIORITY := 7;
   INITIAL_CEILING :     constant PRIORITY := 8;
   Load_Table_WCET  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(3.000 / 1000));
   Send_Table_WCET  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
   thread_action_WCET    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(5.100 / 1000));
end Monitoring_RTATT;

--HSATC operations are executed by the OBCS task inside the rendez-vous.
--The OBCS exports one entry for each HSATC provided operation. They are
--   entered by calling user(s).
--LSATC and ASATC operations are executed by the THREAD task.
--The OBCS exports 3 entries for each LSATC or ASATC. One is similar to
--   the HSATC entries, the others are called only by the THREAD task.
--   They have the purpose to manage in mutual exclusion the parameters
--   buffers and the ATC related status variables.
```

```
with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Monitoring_RTATT;
package Monitoring is        -- CYCLIC HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
   ---------------- OBJECT CONTROL STRUCTURE    ----------------
     --------  DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       -- Load_Table constrained by LSATC
       -- Send_Table constrained by ASATC

   task OBCS is
     pragma PRIORITY(Monitoring_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Load_Table;
     -- called by user object(s)
     entry Load_Table_IN_PROGRESS;
     -- called by THREAD task only
     entry Load_Table_PENDING(IS_PENDING : out BOOLEAN);
     -- called by THREAD task only
     entry Send_Table;
     -- called by user object(s)
     entry Send_Table_IN_PROGRESS;
     -- called by THREAD task only
     entry Send_Table_PENDING(IS_PENDING : out BOOLEAN);
     -- called by THREAD task only
     entry END_ATC_EXECUTION;
     -- called by THREAD task only
   end OBCS;
end Monitoring;

with Real_Time; use Real_Time;    -- Alsys time management services
   ---------------- REQUIRED OBJECT : Bus_Handler    ----------------
     ----------   OPERATION(S)   ----------
   -- Put_report
with Bus_Handler;

package body Monitoring is       -- CYCLIC HRT-HOOD object implementation
----------------   INTERNAL DECLARATIONS   ----------------
     ----------   OPERATION(S)   ----------
   procedure thread_action;
   --------   FORWARD OPERATION DECLARATIONS   ----------
   procedure OPCS_Load_Table;
   procedure OPCS_Send_Table;

   ---------------- OBJECT CONTROL STRUCTURE    ----------------
   --|:OBCS_CODE
   --|:END_CODE
   -- declarations to support LSATC and ASATC operations
   -- operation Load_Table, unbuffered
   Load_Table_CALLED : BOOLEAN := FALSE;
   -- operation Send_Table, unbuffered
   Send_Table_CALLED : BOOLEAN := FALSE;

   -- data for ATC management
   ATC_IN_PROGRESS : NATURAL := 0;

   task THREAD is
     pragma PRIORITY( Monitoring_RTATT.INITIAL_PRIORITY);
   end THREAD;

   task body THREAD is
     T    : Real_Time.TIME := Real_Time.CLOCK;
```

```
   ATC_PENDING        : BOOLEAN;
begin
   if T < RTA.SYSTEM_START_UP_TIME then
      T := RTA.SYSTEM_START_UP_TIME;
   end if;
   T := T + Monitoring_RTATT.THREAD_OFFSET(RTA.START_MODE);
   FOREVER:
   loop
      Real_Time.delay_until(T);
      if ATC_IN_PROGRESS > 0 then
         OBCS.Load_Table_PENDING(ATC_PENDING);
         if ATC_PENDING then
            OBCS.Load_Table_IN_PROGRESS;
            OPCS_Load_Table;
            OBCS.END_ATC_EXECUTION;
         else
         OBCS.Send_Table_PENDING(ATC_PENDING);
         if ATC_PENDING then
            OBCS.Send_Table_IN_PROGRESS;
            OPCS_Send_Table;
            OBCS.END_ATC_EXECUTION;
         else
            null; -- should never be executed
         end if;
         end if;
      else
         thread_action;
      end if;
      T := T + Monitoring_RTATT.THREAD_PERIOD(RTA.CURRENT_MODE);
   end loop FOREVER;
end THREAD;

task body OBCS is
begin
   loop
      select
         accept Load_Table do
            -- if not already pending prepare execution,
            -- otherwise over-write previous request.
            if not Load_Table_CALLED then
               Load_Table_CALLED := TRUE;
               ATC_IN_PROGRESS := ATC_IN_PROGRESS + 1;
            end if;
         end Load_Table;
      or
         accept Load_Table_PENDING(IS_PENDING : out BOOLEAN) do
            IS_PENDING := Load_Table_CALLED;
         end Load_Table_PENDING;
      or
         accept Load_Table_IN_PROGRESS do
            Load_Table_CALLED := FALSE;
         end Load_Table_IN_PROGRESS;
      or
         accept Send_Table do
            -- if not already pending prepare execution,
            -- otherwise over-write previous request.
            if not Send_Table_CALLED then
               Send_Table_CALLED := TRUE;
               ATC_IN_PROGRESS := ATC_IN_PROGRESS + 1;
            end if;
         end Send_Table;
      or
         accept Send_Table_PENDING(IS_PENDING : out BOOLEAN) do
            IS_PENDING := Send_Table_CALLED;
```

```
         end Send_Table_PENDING;
      or
         accept Send_Table_IN_PROGRESS do
            Send_Table_CALLED := FALSE;
         end Send_Table_IN_PROGRESS;
      or
         accept END_ATC_EXECUTION do
            if ATC_IN_PROGRESS > 0 then    -- ignore first call (initialisation)
               ATC_IN_PROGRESS := ATC_IN_PROGRESS - 1;
            end if;
         end END_ATC_EXECUTION;
      or
         terminate;
      end select;
    end loop;
  end OBCS;

  --------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

  procedure OPCS_Load_Table is
  --|:OPCS_CODE <Load_Table>
    begin
      for I in 1 .. 3 loop
        Bus_Handler.OBCS.Get_data;
      end loop;
    --|:END_CODE
  end OPCS_Load_Table;

  procedure OPCS_Send_Table is
  --|:OPCS_CODE <Send_Table>
    begin
      EXE_1_MSEC;
      Bus_Handler.OBCS.Put_Table;
    --|:END_CODE
  end OPCS_Send_Table;

  ---------------- OPCS OF UNCONSTRAINED OPERATIONS     ----------------

  procedure thread_action is
  --|:OPCS_CODE <thread_action>
    begin
      for I in 1 .. 3 loop
        Bus_Handler.OBCS.Get_data;
        EXE_1_MSEC;
      end loop;
      Bus_Handler.OBCS.Put_report;
    --|:END_CODE
  end thread_action;
end Monitoring;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package Scheduler_RTATT is
  CEILING   : constant MODE_PRIORITY := (
         others => 5);
  TRANSFORMATION   : constant INTEGER := 0;
  THREAD_PERIOD    : constant MODE_DURATION := (
         others => Real_Time.To_Time_Span(750.000 / 1000));
  THREAD_OFFSET    : constant MODE_DURATION := (
         others => Real_Time.To_Time_Span(0.000 / 1000));
```

```
   THREAD_DEADLINE  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(750.000 / 1000));
   THREAD_PRIORITY  : constant MODE_PRIORITY := (
          others => 2);
   THREAD_IMPORTANCE     : constant MODE_IMPORTANCE := (
          others => hard);
   INITIAL_PRIORITY :    constant PRIORITY := 2;
   INITIAL_CEILING :     constant PRIORITY := 5;
   Mode_change_WCET : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
   Set_schedule_WCET     : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
   thread_action_WCET    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(3.600 / 1000));
end Scheduler_RTATT;

--HSATC operations are executed by the OBCS task inside the rendez-vous.
--The OBCS exports one entry for each HSATC provided operation. They are
--   entered by calling user(s).
--LSATC and ASATC operations are executed by the THREAD task.
--The OBCS exports 3 entries for each LSATC or ASATC. One is similar to
--   the HSATC entries, the others are called only by the THREAD task.
--   They have the purpose to manage in mutual exclusion the parameters
--   buffers and the ATC related status variables.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Scheduler_RTATT;
package Scheduler is         -- CYCLIC HRT-HOOD object implementation
   ----------------OBJECT DESCRIPTION ----------------
   ----------------PROVIDED INTERFACE ----------------
   ----------------OBJECT CONTROL STRUCTURE    ----------------
     --------  DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       --Mode_change constrained by ASATC
       --Set_schedule constrained by LSATC

   task OBCS is
     pragma PRIORITY(Scheduler_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Mode_change;
     -- called by user object(s)
     entry Mode_change_IN_PROGRESS;
     -- called by THREAD task only
     entry Mode_change_PENDING(IS_PENDING : out BOOLEAN);
     -- called by THREAD task only
     entry Set_schedule;
     -- called by user object(s)
     entry Set_schedule_IN_PROGRESS;
     -- called by THREAD task only
     entry Set_schedule_PENDING(IS_PENDING : out BOOLEAN);
     -- called by THREAD task only
     entry END_ATC_EXECUTION;
     -- called by THREAD task only
   end OBCS;
end Scheduler;

with Real_Time; use Real_Time;    -- Alsys time management services
   ----------------REQUIRED OBJECT : Application    ----------------
     ----------   OPERATION(S)   ----------
   --Stop
   --Start
with Application;
```

```
package body Scheduler is        -- CYCLIC HRT-HOOD object implementation
----------------   INTERNAL DECLARATIONS   ----------------
     ----------   OPERATION(S)   ----------
  procedure thread_action;
  --------   FORWARD OPERATION DECLARATIONS   ----------
  procedure OPCS_Mode_change;
  procedure OPCS_Set_schedule;

  ---------------- OBJECT CONTROL STRUCTURE    ----------------
  --|:OBCS_CODE
  --|:END_CODE
  -- declarations to support LSATC and ASATC operations
  -- operation Mode_change, unbuffered
  Mode_change_CALLED : BOOLEAN := FALSE;
  -- operation Set_schedule, unbuffered
  Set_schedule_CALLED : BOOLEAN := FALSE;

  -- data for ATC management
  ATC_IN_PROGRESS : NATURAL := 0;

  task THREAD is
    pragma PRIORITY( Scheduler_RTATT.INITIAL_PRIORITY);
  end THREAD;

  task body THREAD is
    T    : Real_Time.TIME := Real_Time.CLOCK;
    ATC_PENDING        : BOOLEAN;
  begin
    if T < RTA.SYSTEM_START_UP_TIME then
      T := RTA.SYSTEM_START_UP_TIME;
    end if;
    T := T + Scheduler_RTATT.THREAD_OFFSET(RTA.START_MODE);
    FOREVER:
    loop
      Real_Time.delay_until(T);
      if ATC_IN_PROGRESS > 0 then
        OBCS.Mode_change_PENDING(ATC_PENDING);
        if ATC_PENDING then
          OBCS.Mode_change_IN_PROGRESS;
          OPCS_Mode_change;
          OBCS.END_ATC_EXECUTION;
        else
        OBCS.Set_schedule_PENDING(ATC_PENDING);
        if ATC_PENDING then
          OBCS.Set_schedule_IN_PROGRESS;
          OPCS_Set_schedule;
          OBCS.END_ATC_EXECUTION;
        else
          null; -- should never be executed
        end if;
        end if;
      else
        thread_action;
      end if;
      T := T + Scheduler_RTATT.THREAD_PERIOD(RTA.CURRENT_MODE);
    end loop FOREVER;
  end THREAD;

  task body OBCS is
  begin
    loop
      select
        accept Mode_change do
          -- if not already pending prepare execution,
```

```
          -- otherwise over-write previous request.
          if not Mode_change_CALLED then
            Mode_change_CALLED := TRUE;
            ATC_IN_PROGRESS := ATC_IN_PROGRESS + 1;
          end if;
        end Mode_change;
      or
        accept Mode_change_PENDING(IS_PENDING : out BOOLEAN) do
          IS_PENDING := Mode_change_CALLED;
        end Mode_change_PENDING;
      or
        accept Mode_change_IN_PROGRESS do
          Mode_change_CALLED := FALSE;
        end Mode_change_IN_PROGRESS;
      or
        accept Set_schedule do
          -- if not already pending prepare execution,
          -- otherwise over-write previous request.
          if not Set_schedule_CALLED then
            Set_schedule_CALLED := TRUE;
            ATC_IN_PROGRESS := ATC_IN_PROGRESS + 1;
          end if;
        end Set_schedule;
      or
        accept Set_schedule_PENDING(IS_PENDING : out BOOLEAN) do
          IS_PENDING := Set_schedule_CALLED;
        end Set_schedule_PENDING;
      or
        accept Set_schedule_IN_PROGRESS do
          Set_schedule_CALLED := FALSE;
        end Set_schedule_IN_PROGRESS;
      or
        accept END_ATC_EXECUTION do
          if ATC_IN_PROGRESS > 0 then    -- ignore first call (initialisation)
            ATC_IN_PROGRESS := ATC_IN_PROGRESS - 1;
          end if;
        end END_ATC_EXECUTION;
      or
        terminate;
      end select;
    end loop;
  end OBCS;

  --------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

  procedure OPCS_Mode_change is
  --|:OPCS_CODE <Mode_change>
    begin
      EXE_1_MSEC;
      Application.Stop;
    --|:END_CODE
  end OPCS_Mode_change;

  procedure OPCS_Set_schedule is
  --|:OPCS_CODE <Set_schedule>
    begin
      for I in 1 .. 2 loop
        EXE_1_MSEC;
      end loop;
    --|:END_CODE
  end OPCS_Set_schedule;

  ---------------  OPCS OF UNCONSTRAINED OPERATIONS     ----------------
```

```
   procedure thread_action is
   --|:OPCS_CODE <thread_action>
     begin
       for I in 1 .. 3 loop
         EXE_1_MSEC;
       end loop;
       Application.OBCS.Start;
     --|:END_CODE
   end thread_action;
end Scheduler;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package TC_acquisition_RTATT is
   CEILING   : constant MODE_PRIORITY := (
          others => 10);
   TRANSFORMATION   : constant INTEGER := 0;
   THREAD_PERIOD    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(500.000 / 1000));
   THREAD_OFFSET    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(0.000 / 1000));
   THREAD_DEADLINE  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(10.000 / 1000));
   THREAD_PRIORITY  : constant MODE_PRIORITY := (
          others => 9);
   THREAD_IMPORTANCE     : constant MODE_IMPORTANCE := (
          others => hard);
   INITIAL_PRIORITY :    constant PRIORITY := 9;
   INITIAL_CEILING :     constant PRIORITY := 10;
   Mode_change_WCET : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
   thread_action_WCET    : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(1.000 / 1000));
end TC_acquisition_RTATT;

--HSATC operations are executed by the OBCS task inside the rendez-vous.
--The OBCS exports one entry for each HSATC provided operation. They are
--   entered by calling user(s).
--LSATC and ASATC operations are executed by the THREAD task.
--The OBCS exports 3 entries for each LSATC or ASATC. One is similar to
--   the HSATC entries, the others are called only by the THREAD task.
--   They have the purpose to manage in mutual exclusion the parameters
--   buffers and the ATC related status variables.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with TC_acquisition_RTATT;
package TC_acquisition is        -- CYCLIC HRT-HOOD object implementation
   ---------------- OBJECT DESCRIPTION ----------------
   ---------------- PROVIDED INTERFACE ----------------
   ---------------- OBJECT CONTROL STRUCTURE    ----------------
     -------- DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       --Mode_change constrained by ASATC

   task OBCS is
     pragma PRIORITY(TC_acquisition_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Mode_change;
     -- called by user object(s)
```

```
     entry Mode_change_IN_PROGRESS;
     -- called by THREAD task only
     entry Mode_change_PENDING(IS_PENDING : out BOOLEAN);
     -- called by THREAD task only
     entry END_ATC_EXECUTION;
     -- called by THREAD task only
   end OBCS;
end TC_acquisition;

with Real_Time; use Real_Time;    -- Alsys time management services
   ----------------REQUIRED OBJECT : TC_processing  ----------------
     ----------   OPERATION(S)   ----------
   --Start
with TC_processing;
   ----------------REQUIRED OBJECT : Telecommand    ----------------
     -----------   CONSTANT(S)   ----------
   --The_list
     -------------   TYPE(S)   ------------
   --List
   --Index
   --Item
with Telecommand;

package body TC_acquisition is        -- CYCLIC HRT-HOOD object implementation
----------------   INTERNAL DECLARATIONS   ----------------
     ----------   OPERATION(S)   ----------
   procedure thread_action;
     --------------   DATA   --------------
   Current_Index :  Telecommand.Index := Telecommand.Index'first;
   Current_Item :  Telecommand.Item;
   --------   FORWARD OPERATION DECLARATIONS   ----------
   procedure OPCS_Mode_change;

   ----------------OBJECT CONTROL STRUCTURE    ----------------
   --|:OBCS_CODE
   --|:END_CODE
   -- declarations to support LSATC and ASATC operations
   -- operation Mode_change, unbuffered
   Mode_change_CALLED : BOOLEAN := FALSE;

   -- data for ATC management
   ATC_IN_PROGRESS : NATURAL := 0;

   task THREAD is
     pragma PRIORITY( TC_acquisition_RTATT.INITIAL_PRIORITY);
   end THREAD;

   task body THREAD is
     T    : Real_Time.TIME := Real_Time.CLOCK;
     ATC_PENDING        : BOOLEAN;
   begin
     if T < RTA.SYSTEM_START_UP_TIME then
       T := RTA.SYSTEM_START_UP_TIME;
     end if;
     T := T + TC_acquisition_RTATT.THREAD_OFFSET(RTA.START_MODE);
     FOREVER:
     loop
       Real_Time.delay_until(T);
       if ATC_IN_PROGRESS > 0 then
         OBCS.Mode_change_PENDING(ATC_PENDING);
         if ATC_PENDING then
           OBCS.Mode_change_IN_PROGRESS;
           OPCS_Mode_change;
           OBCS.END_ATC_EXECUTION;
```

```
        else
           null; -- should never be executed
        end if;
     else
        thread_action;
     end if;
     T := T + TC_acquisition_RTATT.THREAD_PERIOD(RTA.CURRENT_MODE);
  end loop FOREVER;
end THREAD;

task body OBCS is
begin
   loop
     select
        accept Mode_change do
           -- if not already pending prepare execution,
           -- otherwise over-write previous request.
           if not Mode_change_CALLED then
              Mode_change_CALLED := TRUE;
              ATC_IN_PROGRESS := ATC_IN_PROGRESS + 1;
           end if;
        end Mode_change;
     or
        accept Mode_change_PENDING(IS_PENDING : out BOOLEAN) do
           IS_PENDING := Mode_change_CALLED;
        end Mode_change_PENDING;
     or
        accept Mode_change_IN_PROGRESS do
           Mode_change_CALLED := FALSE;
        end Mode_change_IN_PROGRESS;
     or
        accept END_ATC_EXECUTION do
           if ATC_IN_PROGRESS > 0 then    -- ignore first call (initialisation)
              ATC_IN_PROGRESS := ATC_IN_PROGRESS - 1;
           end if;
        end END_ATC_EXECUTION;
     or
        terminate;
     end select;
  end loop;
end OBCS;

--------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

procedure OPCS_Mode_change is
--|:OPCS_CODE <Mode_change>
   begin
     EXE_1_MSEC;
   --|:END_CODE
end OPCS_Mode_change;

----------------  OPCS OF UNCONSTRAINED OPERATIONS     ----------------

procedure thread_action is
--|:OPCS_CODE <thread_action>
     use Telecommand;
   begin
     EXE_1_MSEC;
     Current_Item := Telecommand.The_List(Current_Index);
     TC_processing.OBCS.Start(Current_Item);
     if Current_Index < Telecommand.Index'last then
        Current_Index := Current_Index + 1;
     else
        Current_Index := Telecommand.Index'first;
```

```
       end if;
     --|:END_CODE
   end thread_action;
end TC_acquisition;

--It defines the Hard Real-Time attributes of the object.
--All time values are in seconds.

with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
package TC_processing_RTATT is
   CEILING   : constant MODE_PRIORITY := (
          others => 11);
   THREAD_MAT  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(500.000 / 1000));
   THREAD_DEADLINE  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(400.000 / 1000));
   THREAD_PRIORITY  : constant MODE_PRIORITY := (
          others => 4);
   THREAD_IMPORTANCE     : constant MODE_IMPORTANCE := (
          others => hard);
   INITIAL_PRIORITY :    constant PRIORITY := 4;
   INITIAL_CEILING :     constant PRIORITY := 11;
   Start_WCET  : constant MODE_DURATION := (
          others => Real_Time.To_Time_Span(10.000 / 1000));
end TC_processing_RTATT;

--HSATC operations are executed by the OBCS task inside the rendez-vous.
--The OBCS exports one entry for each HSATC provided operation. They are
--   entered by calling user(s).
--LSATC and ASATC operations are executed by the THREAD task.
--The OBCS exports 3 entries for each LSATC or ASATC. One is similar to
--   the HSATC entries, the others are called only by the THREAD task.
--   They have the purpose to manage in mutual exclusion the parameters
--   buffers and the ATC related status variables.


with SYSTEM; use SYSTEM;
with RTA; use RTA;
with Real_Time; use Real_Time;    -- Alsys time management services
with TC_processing_RTATT;
   ----------------REQUIRED OBJECT : Telecommand    ----------------
     -------------   TYPE(S)   ------------
   --Item
with Telecommand;
package TC_processing is         -- SPORADIC HRT-HOOD object implementation
   ----------------OBJECT DESCRIPTION ----------------
   ----------------PROVIDED INTERFACE ----------------
     ----   TYPES FOR CONSTRAINED OPS  ----
   type Start_PARAMETER_SET is record
     TC :  Telecommand.Item;
     OVERRUN :  BOOLEAN := FALSE;
     Not_used  :    Real_Time.Time_Span;
   end record;
   ----------------OBJECT CONTROL STRUCTURE    ----------------
     --------  DESCRIPTION   --------
     ----   CONSTRAINED OPERATION(S)   ----
       --Start constrained by ASER

   task OBCS is
     pragma PRIORITY(TC_processing_RTATT.INITIAL_CEILING);
     pragma Passive;
     entry Start(TC : Telecommand.Item);
```

```
     -- called by user object(s)
     entry WAIT_Start(THE_PARAMS : out Start_PARAMETER_SET);
     -- called by THREAD task only
   end OBCS;
end TC_processing;

   ----------------REQUIRED OBJECT : AOCS_acquisition    ----------------
     ----------   OPERATION(S)   ----------
   --Mode_change
with AOCS_acquisition;
   ----------------REQUIRED OBJECT : Scheduler ----------------
     ----------   OPERATION(S)   ----------
   --Mode_change
   --Set_schedule
with Scheduler;
   ----------------REQUIRED OBJECT : Monitoring----------------
     ----------   OPERATION(S)   ----------
   --Send_Table
   --Load_Table
with Monitoring;
   ----------------REQUIRED OBJECT : Bus_Handler    ----------------
     ----------   OPERATION(S)   ----------
   --Put_report
with Bus_Handler;
   ----------------REQUIRED OBJECT : TC_acquisition ----------------
     ----------   OPERATION(S)   ----------
   --Mode_change
with TC_acquisition;

package body TC_processing is         -- SPORADIC HRT-HOOD object implementation
   --------   FORWARD OPERATION DECLARATIONS   ----------
   procedure OPCS_Start(TC : Telecommand.Item ; OVERRUN : in BOOLEAN ; Not_used : in
Real_Time.Time_Span);

   ----------------OBJECT CONTROL STRUCTURE    ----------------
   --|:OBCS_CODE
   --|:END_CODE
   -- operation Start, unbuffered
   Start_PARAMETERS : Start_PARAMETER_SET;
   Start_CALLED : BOOLEAN := FALSE;

   task THREAD is
     pragma PRIORITY( TC_processing_RTATT.INITIAL_PRIORITY);
   end THREAD;

   task body THREAD is
     Start_BUFFER  :    Start_PARAMETER_SET;
   begin
     FOREVER:
     loop
       OBCS.WAIT_Start(Start_BUFFER);
         OPCS_Start(Start_BUFFER.TC,
                    Start_BUFFER.OVERRUN,
                    Start_BUFFER.Not_used);
     end loop FOREVER;
   end THREAD;

   task body OBCS is
   begin
     loop
       select
         accept Start(TC : Telecommand.Item) do
           -- store parameters
           Start_PARAMETERS := (TC, FALSE, Real_Time.Time_Span_Zero);
```

```
            -- if not already pending prepare execution,
            -- otherwise over-write previous request.
            if not Start_CALLED then
              Start_CALLED := TRUE;
            else
              Start_PARAMETERS.OVERRUN := TRUE;
            end if;
          end Start;
        or
        when Start_Called =>
          accept WAIT_Start(THE_PARAMS : out Start_PARAMETER_SET) do
            THE_PARAMS := Start_PARAMETERS;
            Start_PARAMETERS.OVERRUN := FALSE;
            Start_CALLED := FALSE;
          end WAIT_Start;
        or
          terminate;
        end select;
      end loop;
    end OBCS;

    --------------   OPCS OF CONSTRAINED OPERATIONS   ----------------

    procedure OPCS_Start(TC : Telecommand.Item ; OVERRUN : in BOOLEAN ; Not_used : in
Real_Time.Time_Span) is
    --|:OPCS_CODE <Start in Telecommand>
        use Telecommand;
      begin
        for I in 1 .. 10 loop
          EXE_1_MSEC;
        end loop;
        case TC.OP_code is
          when 1 => Scheduler.OBCS.Set_schedule;
          when 2 => Monitoring.OBCS.Load_Table;
          when 3 => Monitoring.OBCS.Send_Table;
          when 4 => Bus_Handler.OBCS.Put_Report;
          when 5 => AOCS_acquisition.OBCS.Mode_change;
          when 6 => TC_acquisition.OBCS.Mode_change;
          when 7 => Scheduler.OBCS.Mode_change;
          when 8 => null;
          when 9 => null;
          when 10 => null;
          when others => null;
        end case;
      --|:END_CODE
    end OPCS_Start;
end TC_processing;
```