The Olympus Attitude and Orbital Control System†

A Case Study in Hard Real-time System Design and Implementation

A. Burns and A.J. Wellings

Real-time and Distributed Systems Research Group,
Department of Computer Science, University of York, UK

C.M. Bailey and E. Fyfe

British Aerospace Space Systems Ltd,
Communication Satellites Division, Stevenage, UK

ABSTRACT

This paper describes the details of, and the experiences gained from, a case study undertaken by the authors on the design and re-implementation of the Olympus Satellite's Attitude and Orbital Control Systems (AOCS). The goal of the study was to demonstrate that real-time systems can be implemented using Ada and its tasking facilities. The system was designed using HRT-HOOD, analysed using Deadline Monotonic Scheduling Analysis, and implemented on a M68020-based system using a modified York compiler and run-time support system (the modifications are compatible with those proposed for Ada 9X). Our results indicate that systems can be designed to have the flexibility given by multi-tasking solutions, and yet still obtain the same levels of guarantees as those given by cyclic executives.

1. Introduction

Although Ada 83 has made some inroads into the real-time embedded computer systems market, often these systems are programmed in sequential Ada using cyclic executives. Over the last decade much research has been undertaken on the use of process-based systems using preemptive priority scheduling. Techniques such as Rate Monotonic[18] and Deadline Monotonic[17] schedulability analysis are now gaining favour; furthermore the real-time limitations of Ada 83 are well understood[22, 8, 9] and extensive changes have been made in Ada 9X to make the language more responsive to the needs of the real-time community.[16]

This paper describes the details of, and the experiences gained from, a case study undertaken by the authors on the design and re-implementation of the Olympus Satellite's Attitude and Orbital Control Systems (AOCS). The goal of the study[7] was to demonstrate that real-time systems can be implemented using Ada and its tasking facilities (see Locke [19] for a discussion on the advantages of process-based scheduling over cyclic executive). The paper is structured as follows:

- An overview of the system, giving the functional and non-functional requirements.
- The design of the system using the HRT-HOOD design method.[13, 12, 10]
- The implementation of the design in Ada 83 running on top of an augmented stand-alone Ada run-time support system kernel. The kernel has been augmented by those facilities which will be available in Ada 9X.
- A discussion on the problems encountered and how they were solved.

---

† The work has been supported by the European Space Agency (ESTEC Contract 9198/90/NL/SF).

2. The Modelled System: The Olympus AOCS

The Olympus satellite was launched in July 1989 as the world's largest and most powerful civil three-axis-stabilised communications satellite. Situated at longitude 19 degrees West, Olympus provides direct broadcast TV and 'distance learning' experiments to Italy and Northern Europe.

The AOCS subsystem exists to acquire and maintain the desired spacecraft position and orientation. The AOCS software may operate in six modes of operation, of these the Normal Mode is the most complex and is used for the greatest percentage of the satellites lifetime. It is this mode that the study has elected to model.

Hardware Architecture

As depicted in Figure 1, the Normal Mode software is embedded in the Spacecraft Microcomputer Module (SMM) and communicates with the following devices over a serial data bus

- A Telemetry & Telecommand Subsystem (TMTC),
- An InfraRed Earth Sensor (IRES),
- A Digital Sun Sensor (DSS),
- A Rate Gyro Sensor (RGS),
- Four reaction wheels (RWs),
- Thrusters.

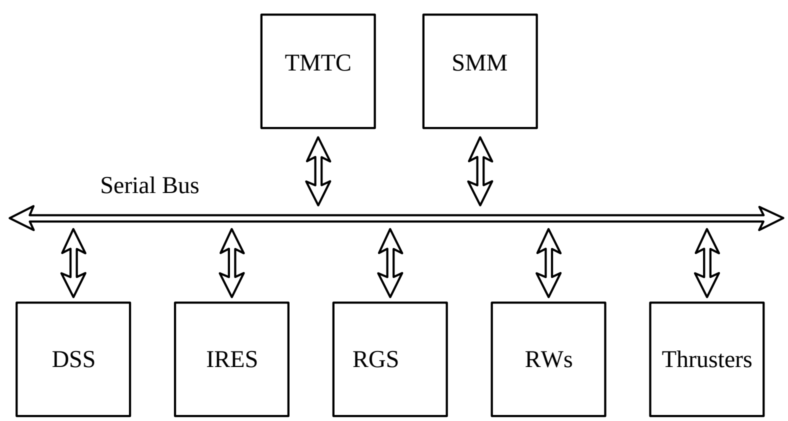


Figure 1: The AOCS Hardware Architecture

There is a cold-standby SMM which is powered up should a failure be detected. In this case study we are interested in real-time aspects of the system, and therefore we shall assume the system's hardware and software is reliable.

Serial bus messages are placed on the bus according to the priority of the transmitting device. Gyros have the highest priority, followed by the TMTC and the SMMs. The bus has time slots reserved for replies, ensuring that SMM requests for sensor data receive responses within a 960 µs time slot. There is a 10ms real-time clock.

Software Requirements

The object of 'Normal Mode' attitude control is to maintain the satellite's orientation to Earth. This is to be achieved by a 200 ms cyclic task that forces IRES roll and pitch angles and a rate-gyro derived yaw angle to zero by controlling the speed of four reaction wheels.

For two spells per day of 15 minutes, a one second period task calibrates the gyro drift rate by comparing the yaw estimate, derived from an integration of gyro rate, against the sun and earth positions. The gyro angle and gyro drift rate are corrected at the end of each spell. The gyro data is received approximately every 100ms (without being requested).

In addition to the regular activities identified above, further attitude control functions occur at less frequent or irregular intervals.

- Momentum dumping is triggered when the speed of any reaction wheel exceeds a preset threshold. This consists of a reduction in the reaction wheel speed in a series of steps, while compensating bursts of thruster firings prevent the loss of Earth-pointing. Dumping on the three axes operates independently.
- The Telemetry and Telecommand Subsystem (TMTC) routinely requests status information from the AOCS software. This task, which has a minimum period of 62.5 ms, keeps the ground informed of the spacecraft state. The SMM is unable to respond to a telemetry request in the same bus time slot, so it transmits a response in a later time slot.
- A telecommand function allows ground to enable or disable control, to enable or disable dumping, to trigger gyro calibration, or to set a reaction wheel failed or operational. Telecommands can occur at a minimum interval of 190 ms.

For a full description of the Requirements see Bailey.[6]

The application selected contains many typical features of embedded real-time space software,[5] namely:

- Cyclic tasks,
- Sporadic tasks,
- Hard real-time tasks, Soft real-time tasks,
- Background tasks,
- Communication over a bus.

Currently, the operational software is coded in 9989 assembler and scheduled by a cyclic scheduler.

3. Software Design

We represent the redesign of the AOCS using HRT-HOOD.[13, 12] HRT-Hood is a new design methodology that builds on the foundations of HOOD.[2] It combines object oriented design and hierarchical decomposition with explicit abstractions which support common hard real-time design paradigms. HRT-HOOD recognises the following object types:

- passive — similar to those in HOOD
- active — similar to those in HOOD
- cyclic — objects which represent periodic activities
- sporadic — objects which represent aperiodic, or sporadic, activities
- protected — objects which control access to resources
- class — similar to those in HOOD
- op_control — similar to those in HOOD
- environment — similar to those in HOOD

Objects are described by their operations, their threads of control, their synchronisation with other objects, and their real-time attributes. For a full description of HRT-HOOD, the reader is referred to the literature.[13, 12]

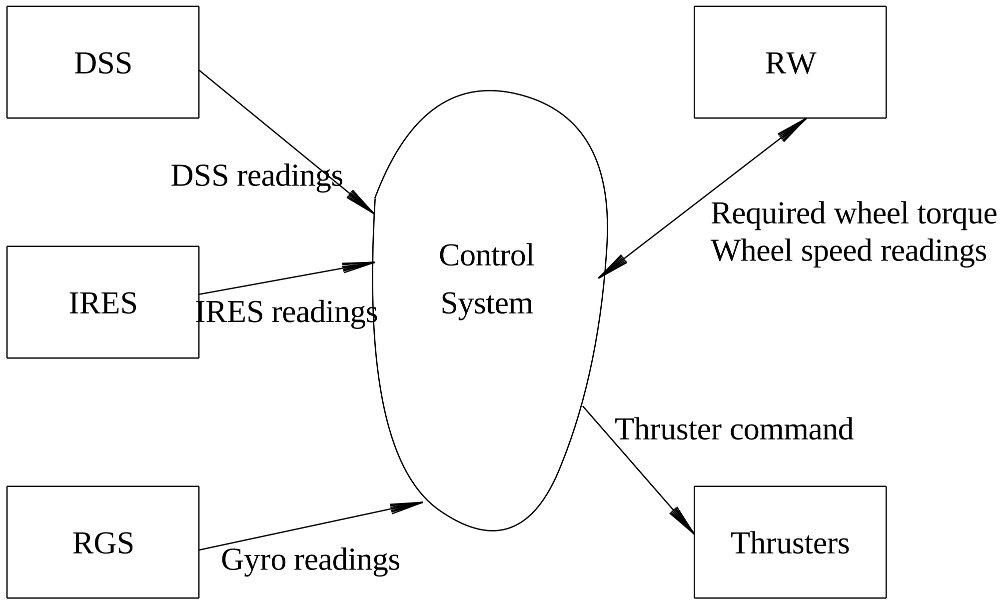


Figure 2: Relationship between External Devices and the Control System

3.1. Relationship between External Devices and the Control System

Figure 2 shows the context in which the control software is to be designed.

It should be noted that all sensor readings and actuator commands are communicated over a serial bus.

3.2. First Level Decomposition

Figure 3 shows the first level decomposition of the software. The software is basically constructed from three subsystems: the CONTROLLER, an active object, which implements the main AOCS software (monitoring sensors, initiating actuators, and implementing the control laws), the interface to the Telemetry and Telecommand Subsystem (which is shown for convenience as two objects: a terminal sporadic object to implements the incoming commands, and an active object which is responsible for sending status reading to ground), and a bus controller subsystems (which again for convenience is shown as two objects: an active object for handling incoming messages, and a terminal protected objects for placing data in the hardware FIFO buffer for output onto the network). The diagram also shows the real-time attributes of the terminal objects which have been added.

3.3. The CONTROLLER Object

The decomposition of the controller object is shown in Figure 4. It consists of:

- several protected objects — which are used to control access to data which is shared between the activities of the system; in particular the SERIAL BUS IP protected object is used to encapsulate the data received from the bus (via the RECEIVE FROM BUS object),
- the "CONTROL_LAW" terminal cyclic object — which implements the basic control laws, and is therefore responsible for maintaining the Satellites Earth point.
- the "SENSOR" active object —— which monitors the satellites sensors and provide information for the CONTROL_LAW object,
- the "ACTUATORS" active object — which controls access to the actuators.

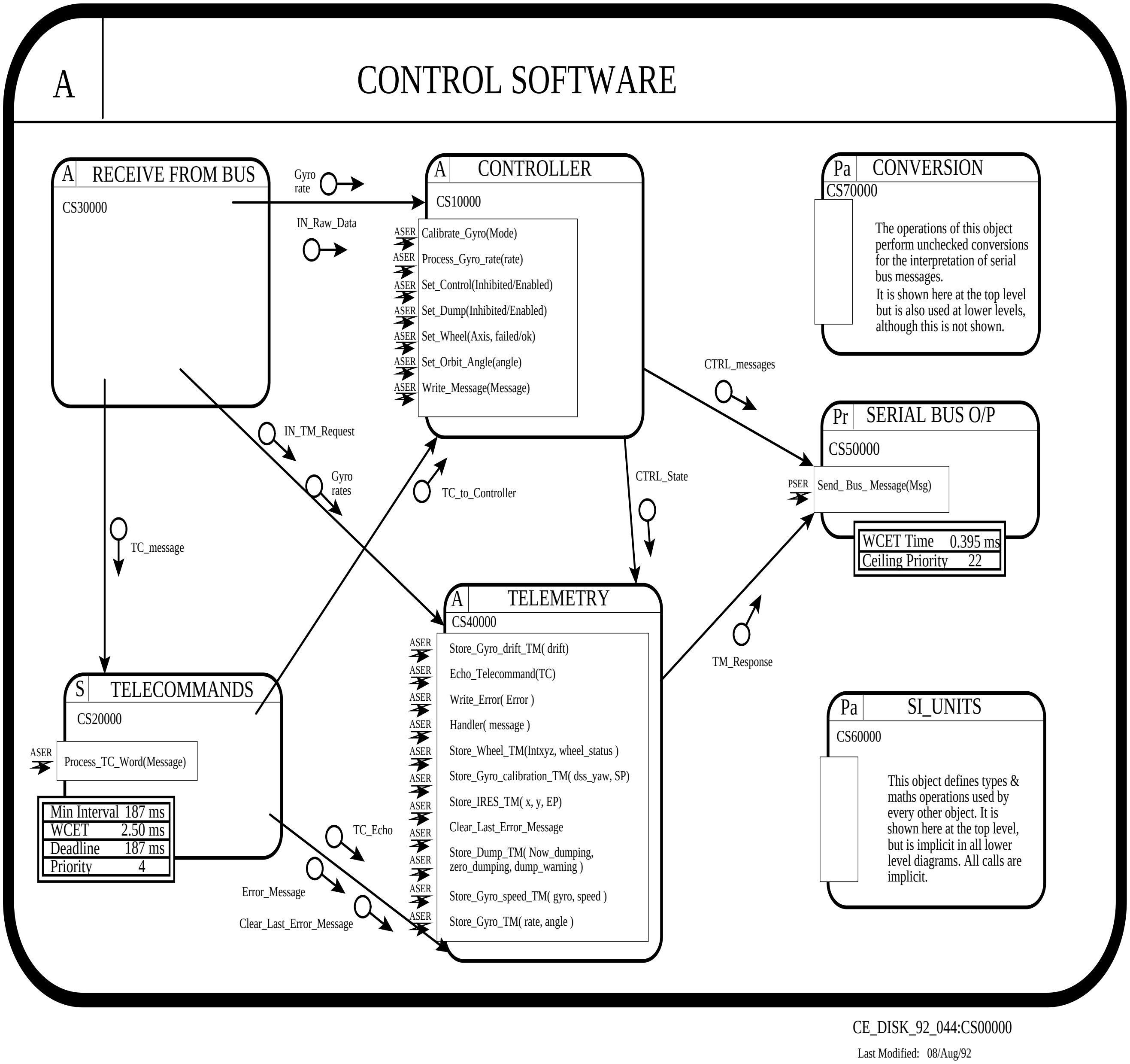


Figure 3: The Control Software

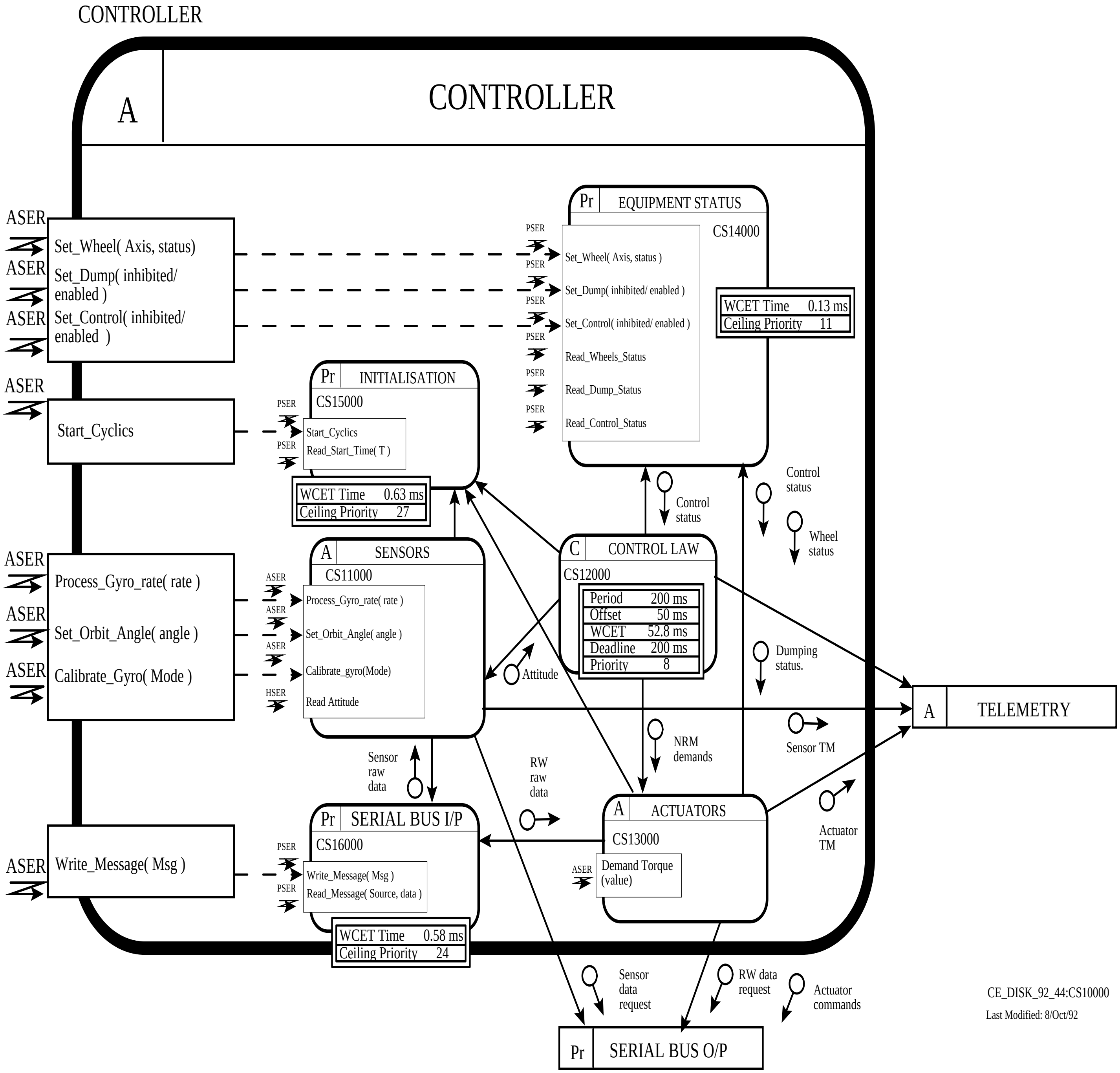


Figure 4: The CONTROLLER Object

Further decomposition of the SENSOR object is shown in Figures 5 to 9.

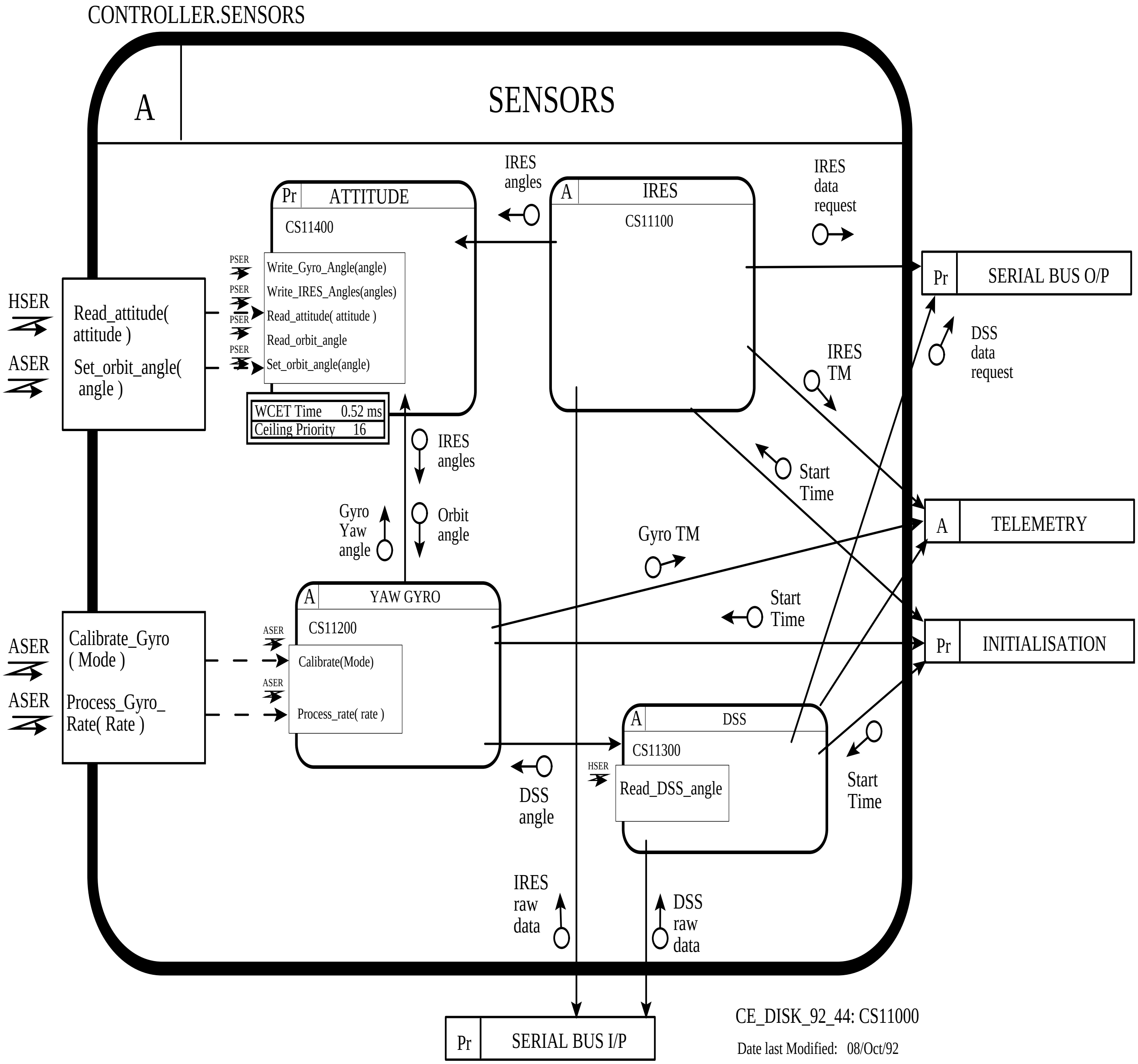


Figure 5: The SENSOR Object

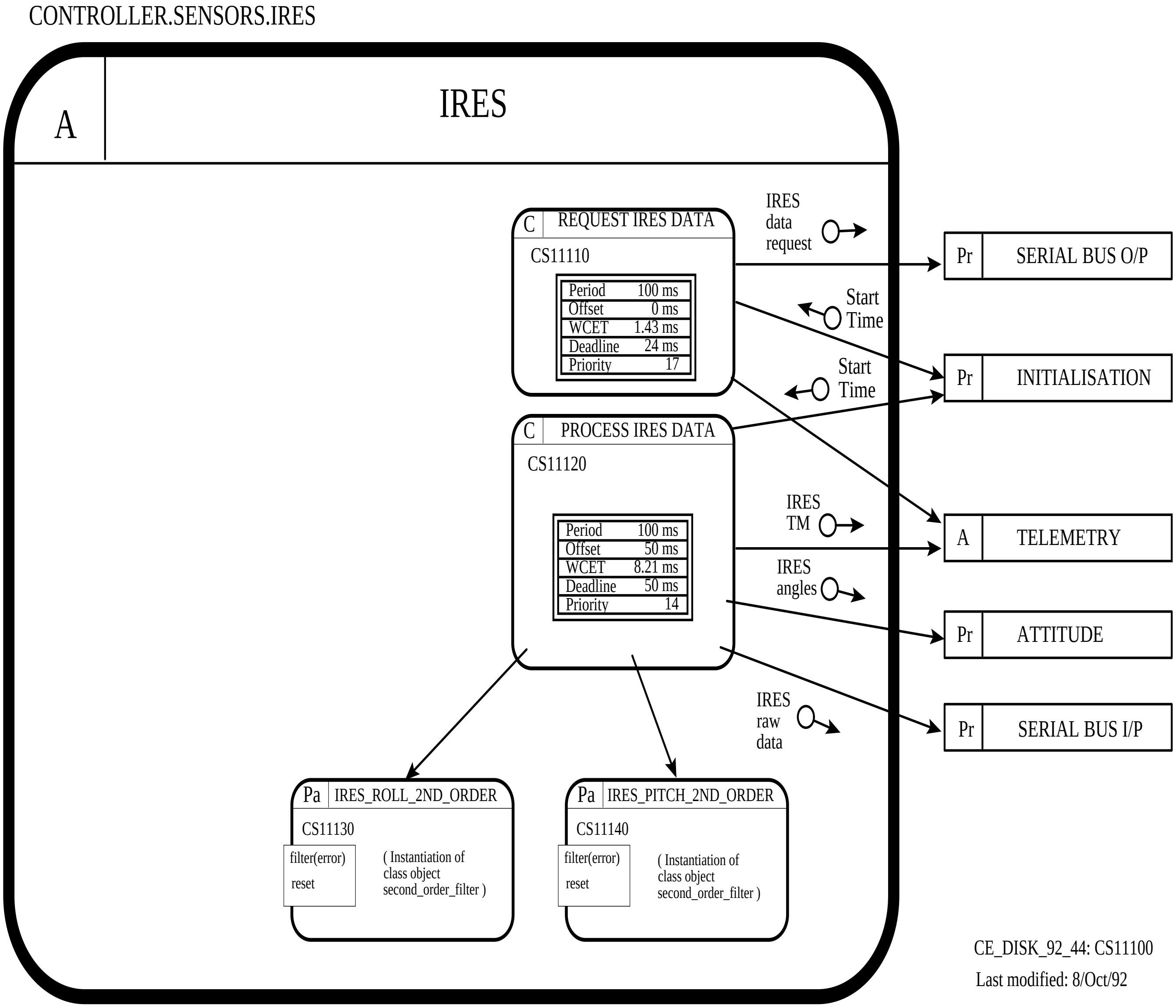


Figure 6: The IRES Object

The IRES sensor controller consists of two precedent constrained cyclic objects with a time offset between the two implementing the required synchronisation. The first object, REQUEST IRES DATA, sends a request to the IRES (via the serial bus). The second object will receive and interpret the sensor values. The relative time offset between the task releases and the deadline of the first object ensures that the sensor device has a chance to respond (at least 30 ms).

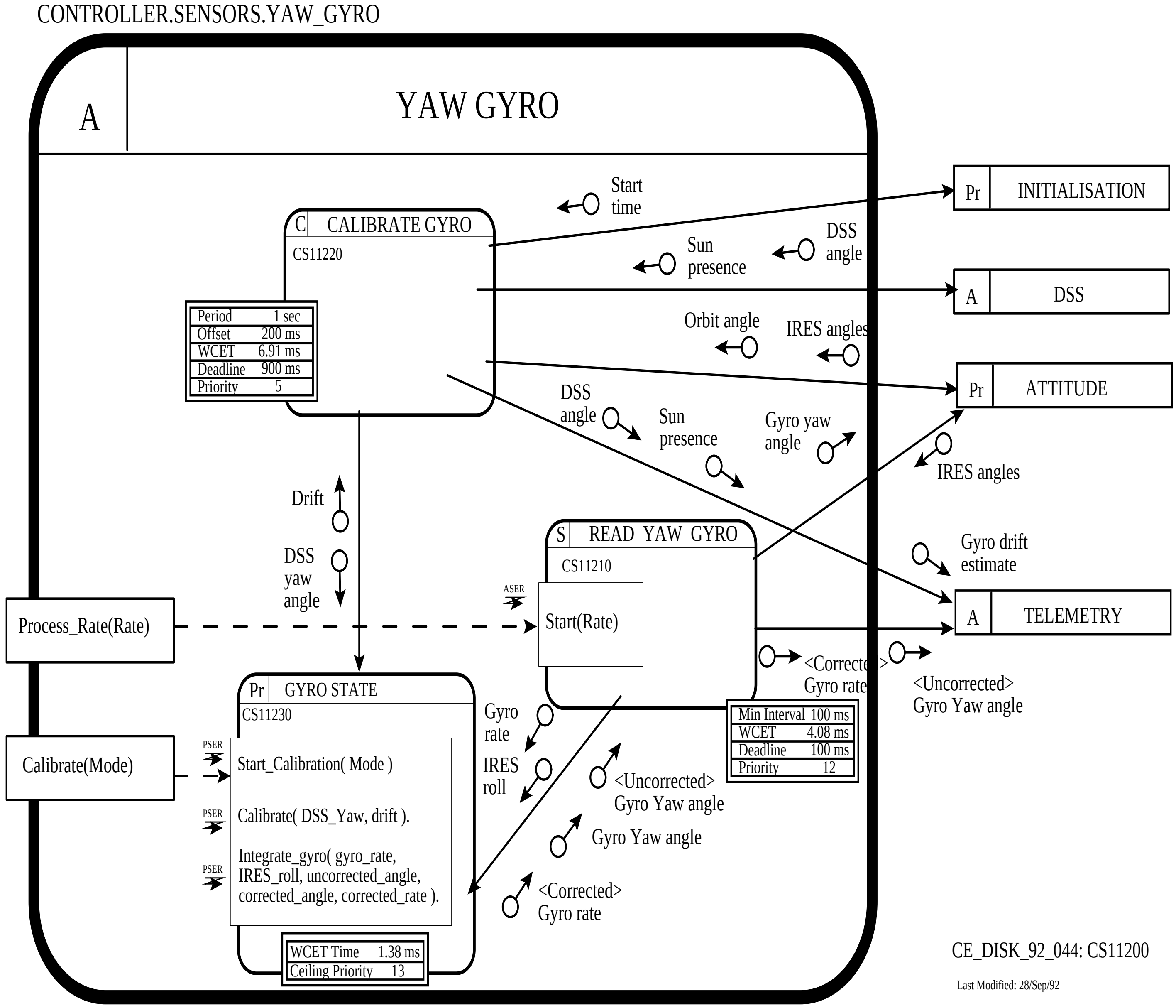


Figure 7: The YAW GYRO Object

The gyro processing consists of cyclic object which calibrates the gyros every second. The sporadic object, READ YAW GYRO, processes the incoming data from the sensors. Note that this object is a sporadic even though the data comes in regularly. This is because the sensor has a different clock and there may be some jitter on the received data.

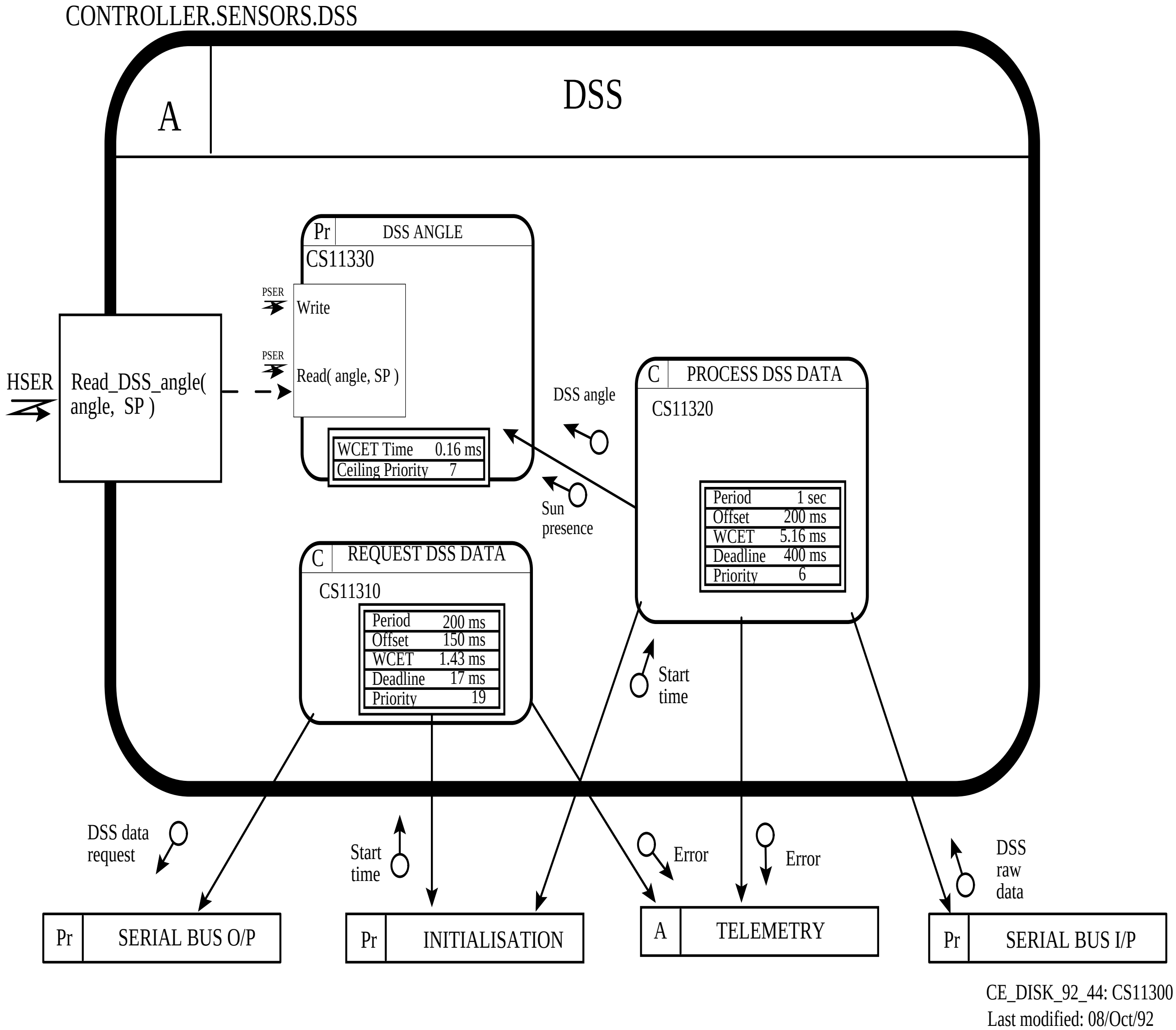


Figure 8: The DSS Object

The digital sun sensor, again, consists of two cyclic objects with a relative offset between them. The deadline of 20ms on the REQUEST DATA object ensures that there are 30 ms available for delivery of the message and for the sensor to respond.

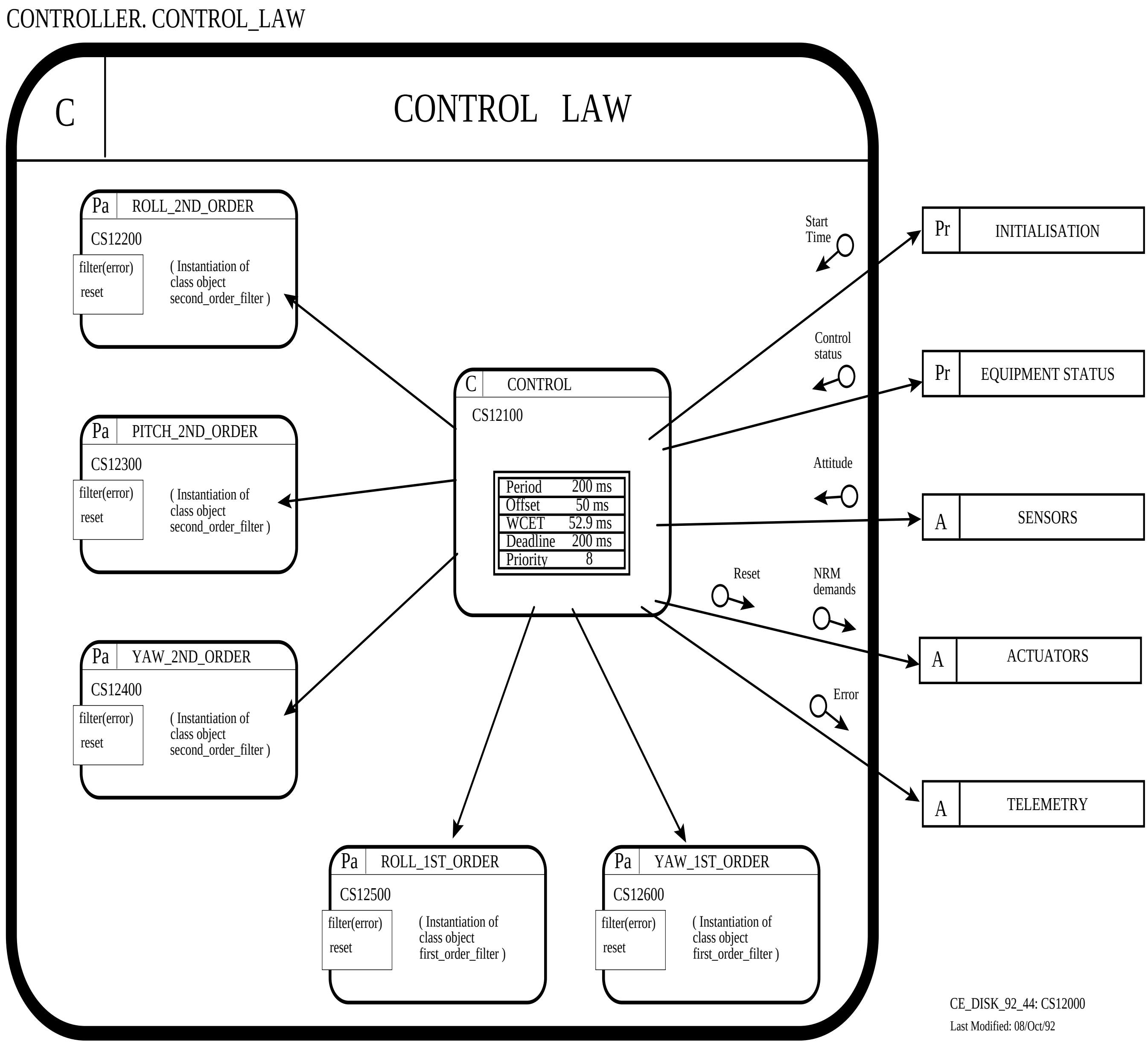


Figure 9: The CONTROL LAW Object

The CONTROL LAW Object consists of a 200 ms cyclic object which implements the basic control laws for the satellite by monitoring the sensors and sending commands to the actuators.

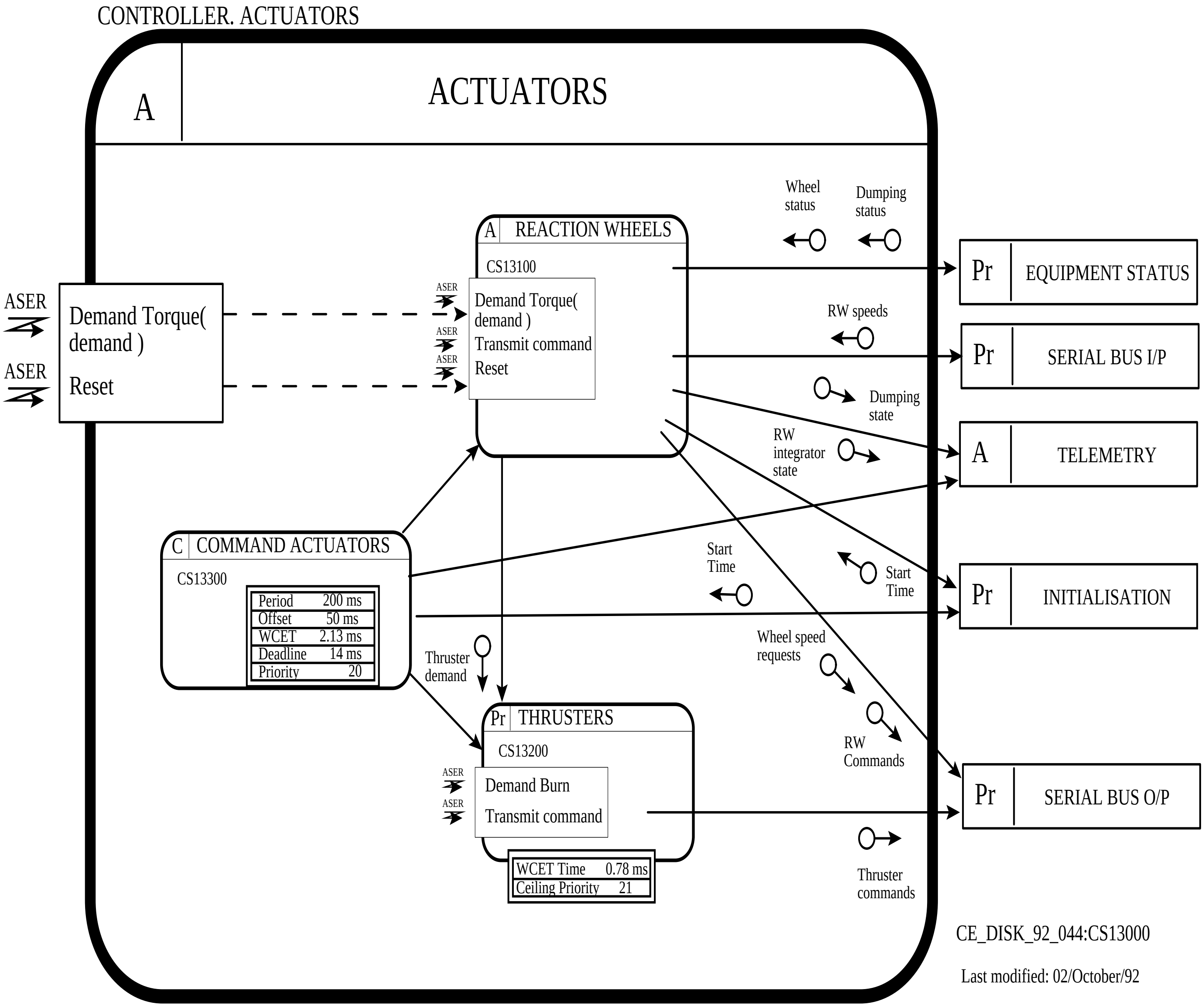


Figure 10: The ACTUATORS Object

The ACTUATOR object encapsulates the reaction wheels and the thruster actuators. The decomposition of the REACTION WHEELS object is shown in Figures 11 to 13.

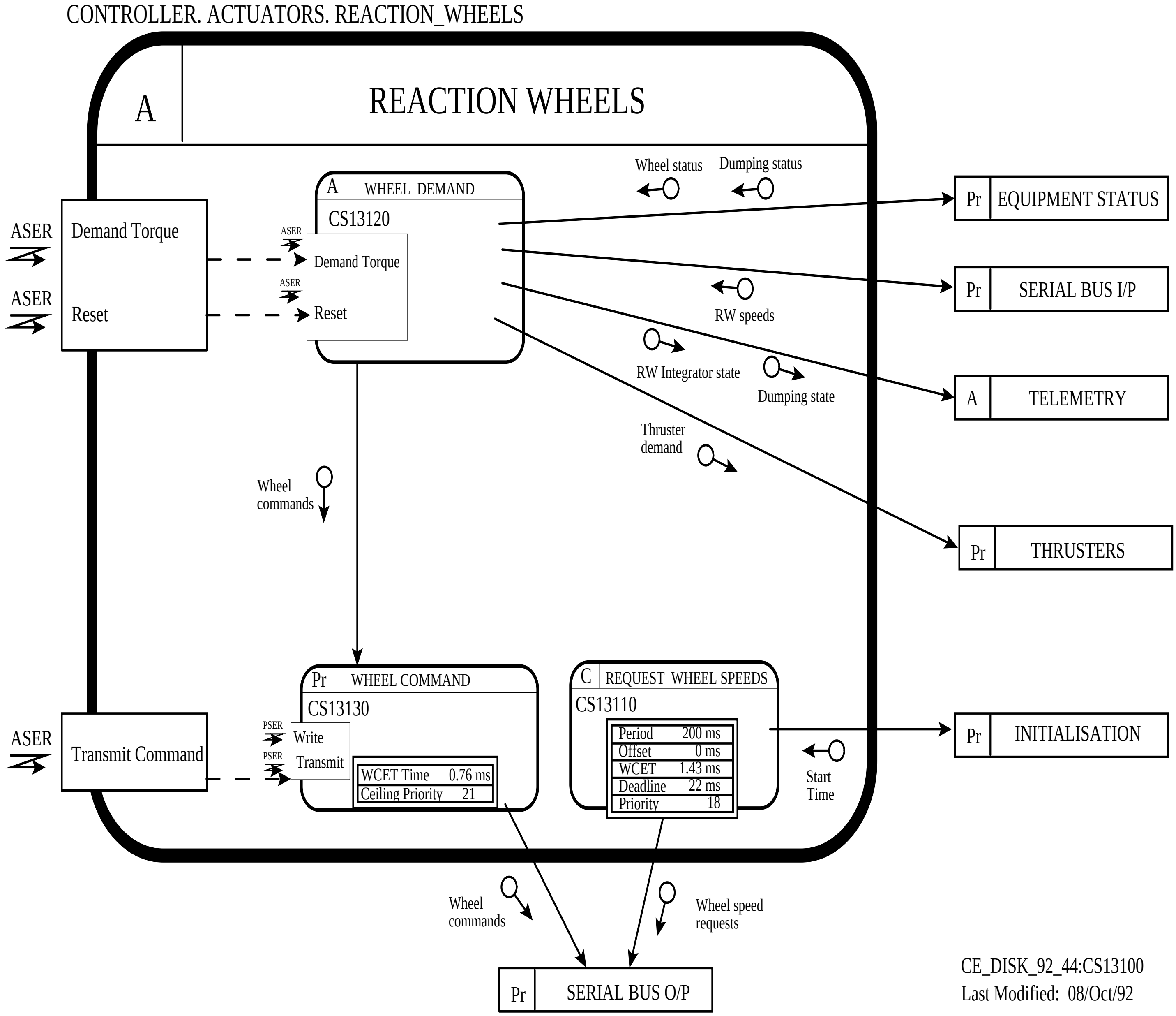


Figure 11: The REACTION WHEELS Object

The REACTION WHEELS object controls the operation of the reaction wheels. Although we consider the wheels to be actuators, they also provide readings giving their current speed. The cyclic object REQUEST WHEEL SPEEDS requests those speeds every 200 ms. The values returned from the device is held in the SERIAL BUS I/P Object.

The WHEEL COMMAND object simply constructs the command for forwarding onto the SERIAL BUS I/O object for transmission across the network.

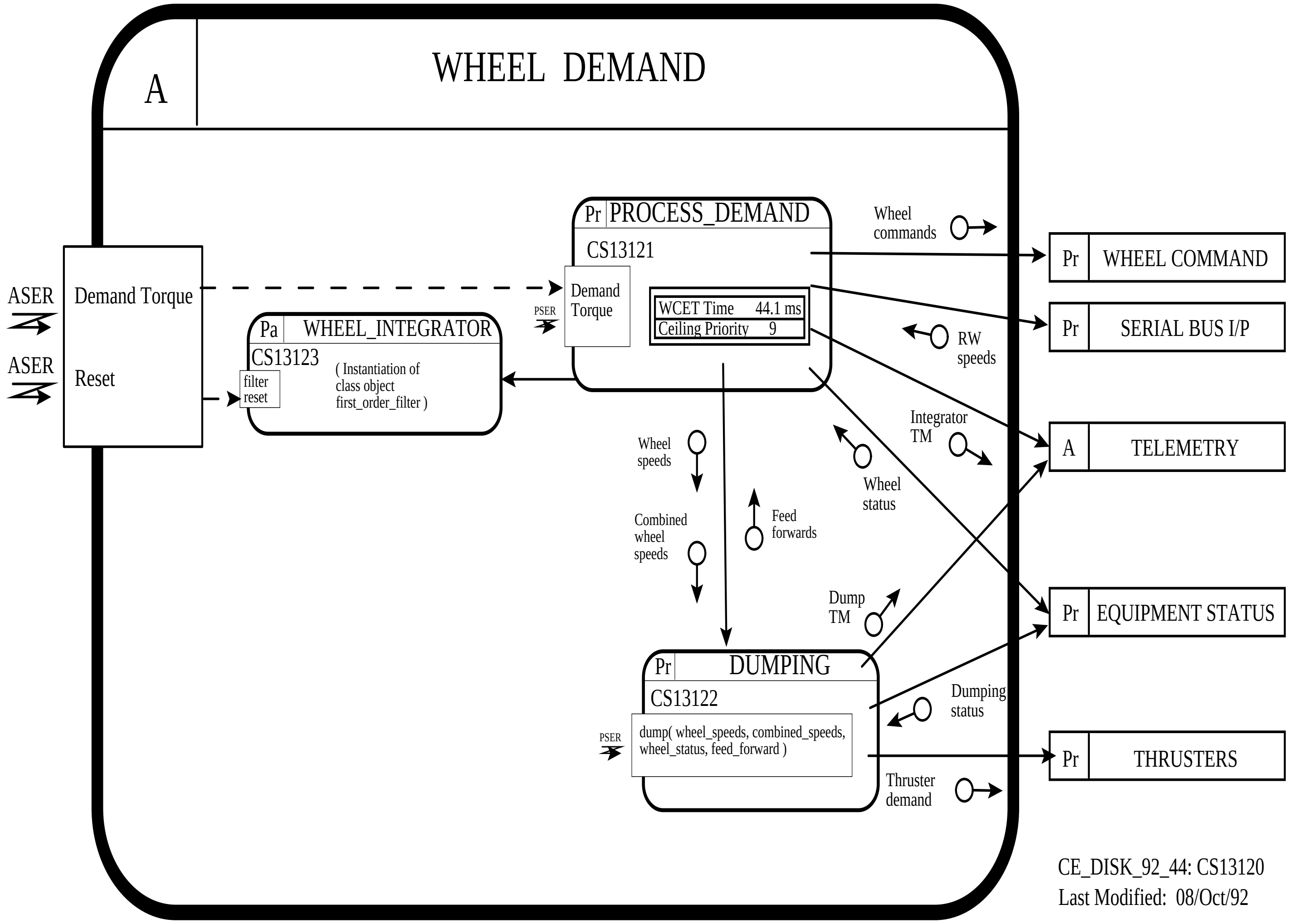
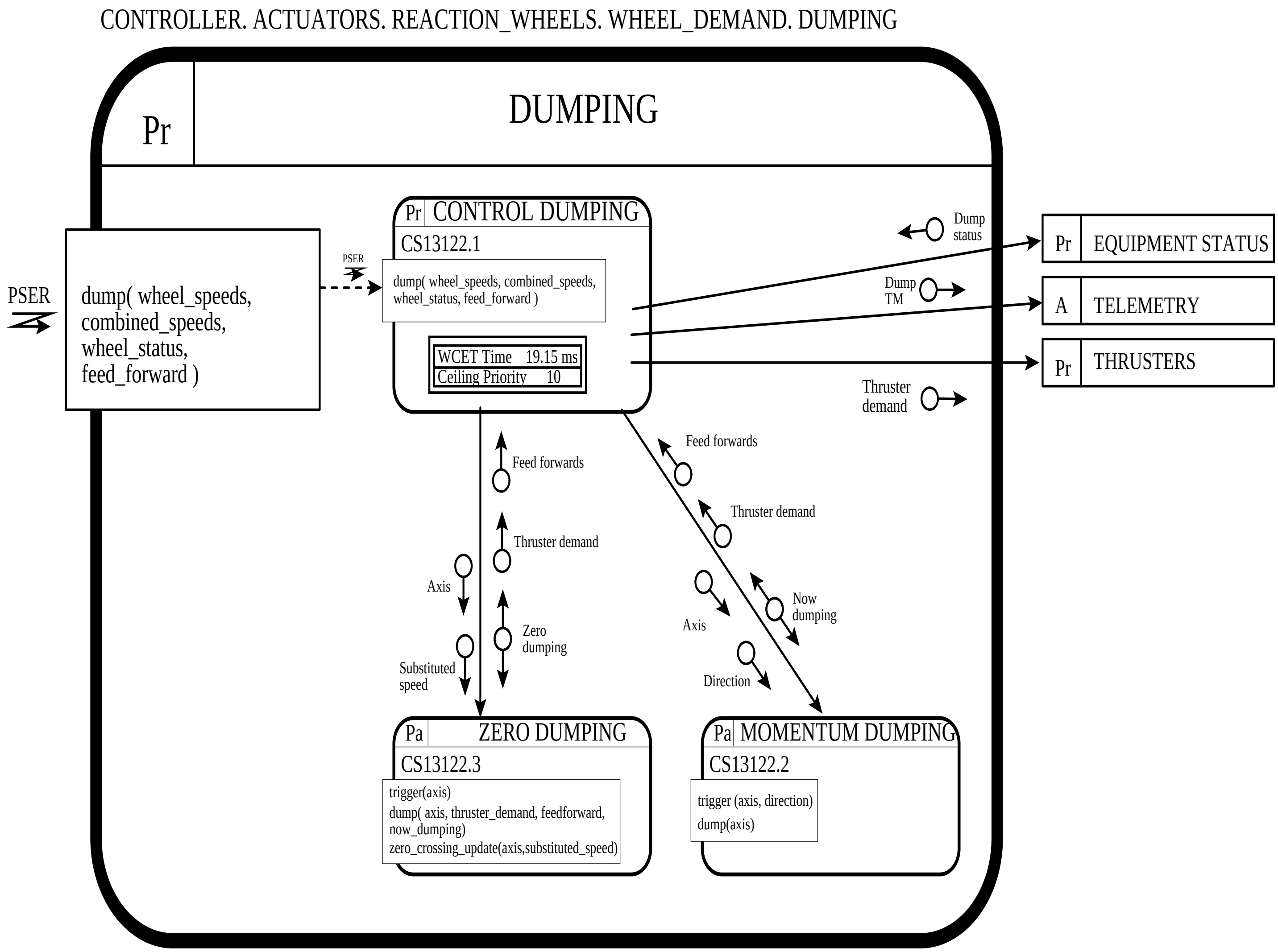


Figure 12: The WHEEL DEMAND Object

The WHEEL DEMAND object provides the functionality for driving the actuator. It consists of two protected object which either issue commands to the wheels or instructs the thrusters to fire.



CE_DISK_92_44: CS13122
Last Modified: 8/Oct/92

Figure 13: The DUMPING Object

3.4. RECEIVE FROM BUS Object

The RECEIVE FROM BUS object handles the input data arriving on the serial bus. A sporadic object responds to the bus interrupt and places the data into a buffer. A cyclic objects then retrieves the data and passes it on to the appropriate receiving object.

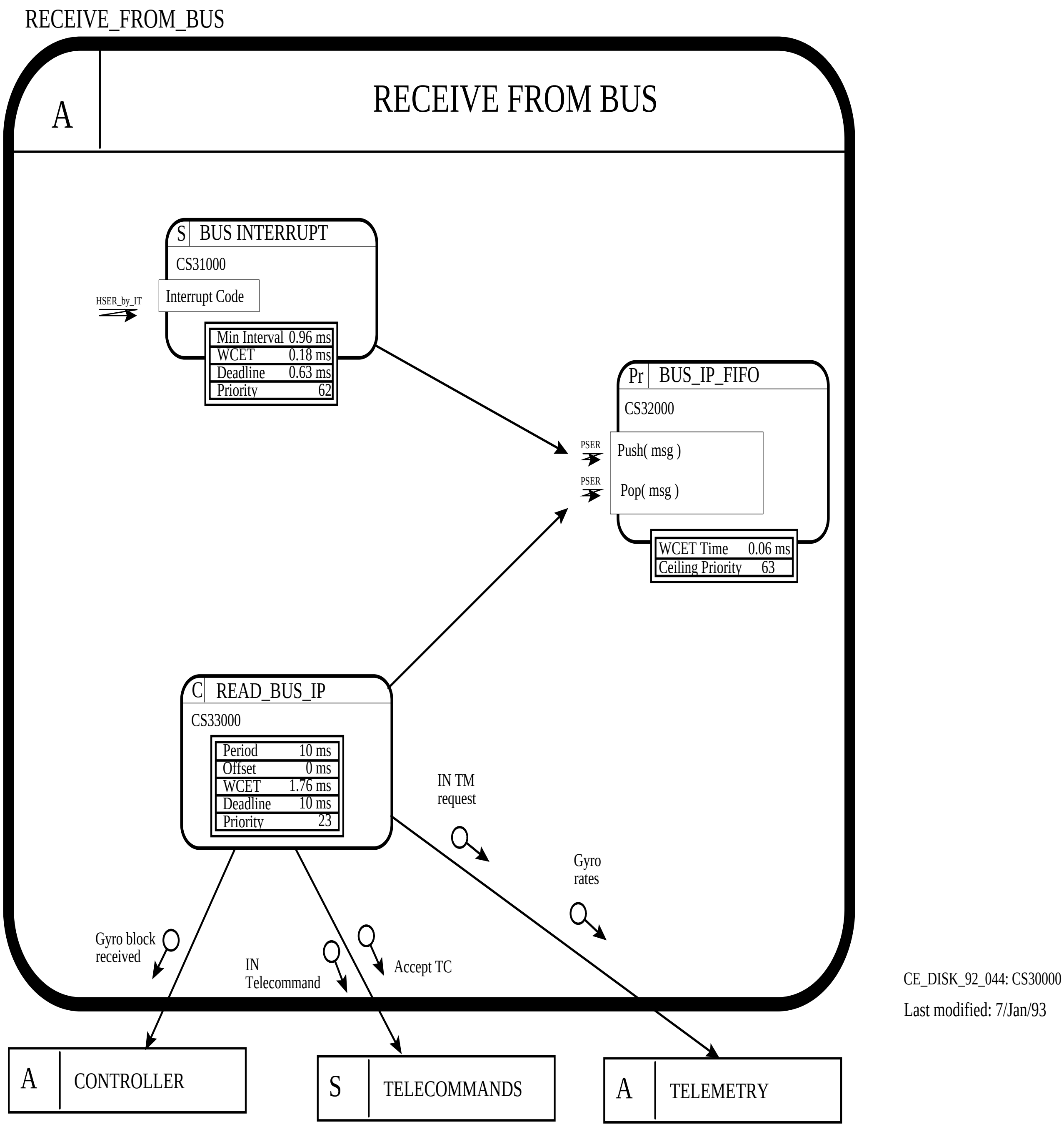


Figure 14: RECEIVE FROM BUS Object

3.5. TELEMETRY Object

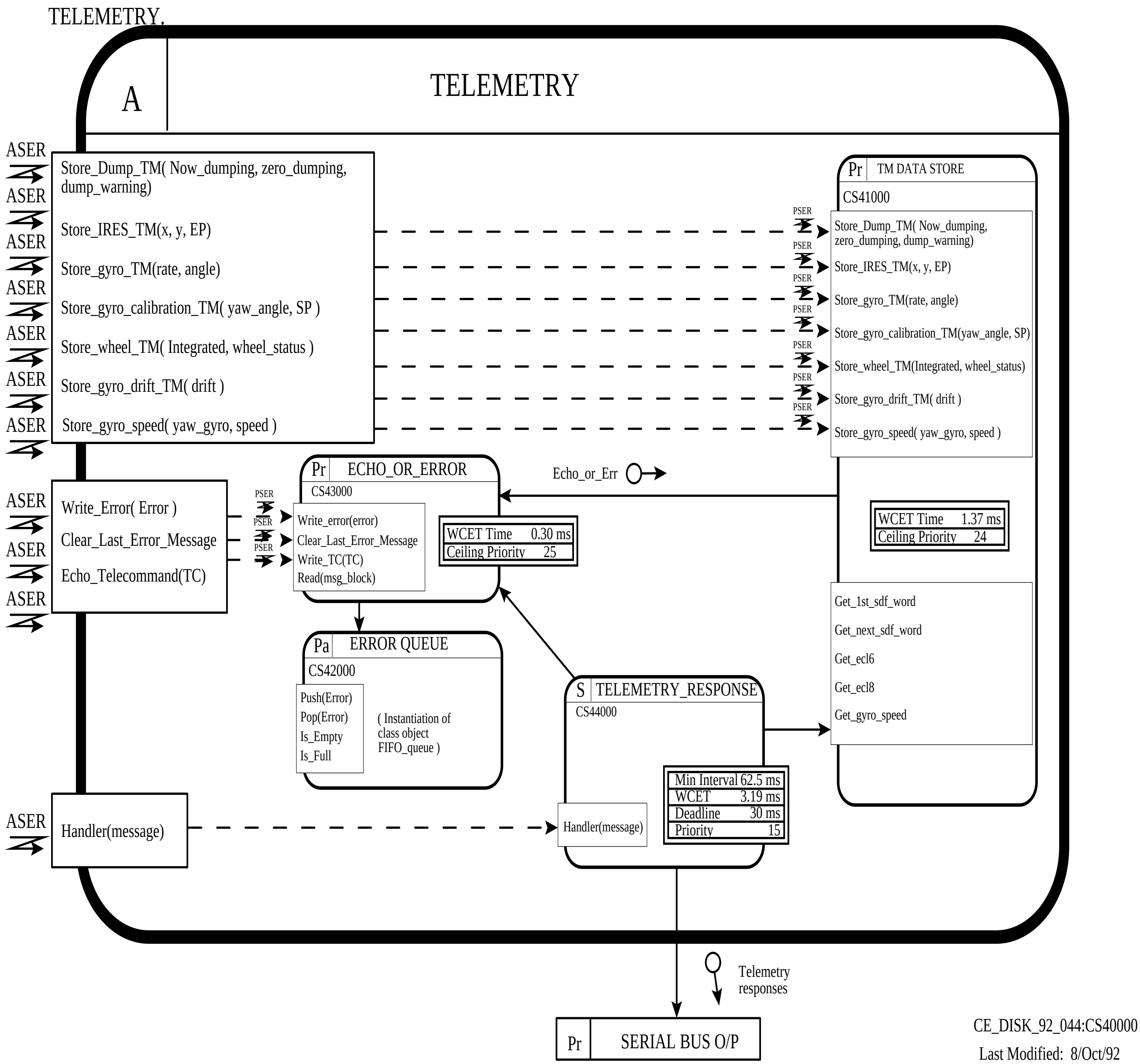


Figure 15: TELEMETRY Object

The TELEMETRY object is responsible for storing sensor readings and passing them to ground when requested.

4. The Execution Environment

This section describes the proposed execution environment for the system and how the software design is mapped to it.

4.1. The Hardware Platform

The case study runs on 2 VME boards containing a 68020 processor, a 68881 floating point coprocessor, 1 MByte RAM, timers, and dedicated chips for communication over the Olympus serial bus.

These new cards replace the current Spacecraft Microcomputer Module (SMM) in the Olympus [AOCS] Engineering Model testbed to demonstrate successful operation of the unit.

For ease of timing analysis, neither the processor cache nor DMA was used.

4.2. The Operating System

The YSE Ada compiler[1] and a modified stand-alone run-time kernel has been used in this study. The system has been modified to support some of the new features proposed for the new Ada 9X standard.

- Large priority range
- Priority queuing
- Protected objects — implemented as optimised passive tasks
- Delay until

For ease of analysis, the following features of Ada have not been used:

- Dynamic task creation and abortion,
- Access types,
- Dynamic memory allocation or deallocation,
- The Ada83 rendezvous (rather protected objects are used).

4.3. Mapping the Software Architecture to the Execution Environment

The final HOOD design containing the following application terminal objects:

- 9 cyclic objects,
- 3 sporadic objects,
- 14 protected objects
- 16 passive objects,

and produced a total of 3300 lines of Ada code (see Appendix 1 for an overview of mapping HRT-HOOD designs to Ada 83 code).

Criticality

The requirement for the AOCS identified two levels of software criticality. The majority of software must be guaranteed to met all deadlines; these are denoted as HARD. The remaining software is non-critical and denoted as SOFT. In the following scheduling analysis, the HARD process are considered first. All SOFT processes are assigned priorities below those used by the HARD.

Schedulability Analysis

Using Deadline Monotonic scheduling analysis[17] each task is given a unique priority (P); the higher the priority the shorter the task's deadline. In the following discussions the task set is assumed to be ordered by priority such that P_1 is the highest priority and P_N the lowest.

To determine whether a given task set is schedulable, it is necessary to calculate the worst case response time of each task in the system (the time at which each task finishes its execution). The worst case response time, R, for each task can be calculated using the following relationships (see Audsley et al[4] for a derivation of these equations). If the method fails to converge to a value of R (or the value obtained is greater than D, the deadline requirement) then the task cannot be guaranteed. The relationships assume the following:

- There exists the possibility that all tasks will be released at the same time (the *critical instant*). If it can be proved that the tasks do not share a critical instant then the predictions made by these relationships are pessimistic.
- Kernel costs such as context switch time are subsumed into the computation time of each thread/process.
- Clock overheads are modelled as a single cyclic task with constant execution time. It should be noted that this is pessimistic and better models are now available.[14]

The basic scheduling equation is simple (for all tasks τ_i):

$$R_i = C_i + B_i + I_i$$

Where B_i is the blocking time that the task experiences, and I_i is the interference τ_i experiences from higher priority tasks.

A task is blocked when it is prevented from running by a lower priority task. For example, if a high priority task shares a critical section with a low priority task then it is possible for the high priority task to be delayed while the other task is actually executing within the critical section. Blocking (priority inversion) can also occur when the lower priority task is executing a non-preemptable kernel routine.

In order to bound the blocking time some form of priority inheritance is needed.[21] In this study we used *immediate ceiling priority inheritance* (also called *ceiling priority emulation*). Critical sections are assigned ceiling priorities. This represents the highest priority of any task that uses that critical section. Whenever a task accesses a critical section its priority is immediately raised to that of the ceiling. As a consequence mutual exclusion (on a single processor) is assured (the current task is running with a priority at least as high as any task that could also wish to enter the critical section). It is also the case that a task is blocked at most once during its execution (the proof of this statement can be found in the literature[21, 20]).

An estimation of I_i is obtained by noting that in any time interval $[0,R_i)$ the maximum load to be asserted by higher priority tasks is

$$\sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Hence we get the relationship:

$$R_i = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

To solve this an interactive approach is used:[4]

$$R_i^n = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{n-1}}{T_j} \right\rceil C_j$$

With R_i^0 equal to C_i

This interaction terminates when either $R_i^{n-1} = R_i^n$ or (unsuccessfully) when $R_i^n > D_i$.

Results of the Analysis

In order to undertake schedulability analysis it is necessary to have the real-time characteristics of the terminal objects. The worst case execution times of the objects were calculated using a tool constructed by the project.[15] Other characteristics, such as cycle times of periodic processes are known from the requirements. The following tables summarise the real-time characteristic of the final system, and gives the task and protected object priorities which were calculated by a tool constructed by the project.[11] All times are in ms. A more detailed description of the results is given in Appendix 2.

Task name	Importance	Period	Offset	WCET	Required Deadline	Achieved Deadline	Priority
RTC	HARD	50	0	0.28	9.0	3.52	27
Read_Bus_IP	HARD	10	0	1.76	10.0	6.99	23
Command_Actuators	HARD	200	50	2.13	14.0	13.52	20
Request_DSS_Data	HARD	200	150	1.43	17.0	15.87	19
Request_Wheel_Speeds	HARD	200	0	1.43	22.0	18.22	18
Request_IRES_data	HARD	100	0	1.43	24.0	23.37	17
Process_IRES_data	HARD	100	50	8.21	50.0	44.13	14
Control_Law	HARD	200	50	52.84	200.0	183.50	8
Process_DSS_Data	HARD	1000	200	5.16	400.0	198.38	6
Calibrate_Gyro	HARD	1000	200	6.91	900.0	389.49	5

Table 1: Cyclic Thread Characteristics

Task name	Importance	Minimum Arrival Time	WCET	Required Deadline	Achieved Deadline	Priority
Bus_Interrupt	INTERRUPT	0.96	0.18	0.63	--	62
Telemetry_Response	HARD	62.5	3.19	30.0	28.73	15
Read_Yaw_Gyro	HARD	100.0	4.08	100.0	55.84	12
Telecommands	SOFT	187.0	2.5	187.0	FAIL	4

Table 2: Sporadic Thread Characteristics

PR name	WCET	Ceiling Priority
Bus_IP_FIFO	0.06	63
Initialisation	0.63	27
Serial_Bus_IP	0.58	24
Read_Yaw_Gyro.OBCS	0.15	24
Telecommands.OBCS	0.15	24
Telemetry_Response.OBCS	0.15	24
Echo_or_Error	0.30	25
Serial_Bus_OP	0.39	22
Wheel_command	0.76	21
Thrusters	0.78	21
TM_data_store	1.37	24
Attitude	0.52	16
Gyro_state	1.38	13
Equipment_status	0.13	11
Control_dumping	19.15	10
Process_demand	44.09	9
DSS_angle	0.16	7

Table 3: Protected Tasks Characteristics

Note that to implement sporadic objects requires a synchronisation agent. This is in fact a form of protected object. There are three of these identified as X_OBCS.

5. Problems Encountered

The main problem we encountered was associated with handling the interrupts off the bus. Originally we attempted to map a sporadic object to the bus interrupt and to have this object pass on the data to the various sensor objects. A sporadic object in HRT-HOOD is mapped to an Ada task and a passive (protected) Ada task (representing an Ada 9X protected object). The passive task handles the interrupt, and releases the other task to deal with the received data. However, the schedulability analysis indicated that with a minimum inter-arrival time of 960 µs for the sporadic (the estimated minimum time between interrupts) the system was not schedulable. In fact the overhead of entering the Ada passive task and releasing the sporadic was almost 960 µs. This reflected the prototype nature of our modifications to Ada to implement the equivalent of an Ada 9X protected record.

The problem was overcome by

1) Modifying the analysis — it was recognised that the minimum inter-arrival rate for the interrupt was not sustained over a long period; the system (for analysis purposes) was more accurately modelled as the sum of four interrupts (representing the four different message types that could be received from the sensors — called Message_here, TM_here, Z1_here, and TC_here in Appendix 2), each with their characteristic minimum interarrival time. These task are not themselves analysed but rather are used to model the interference on other tasks.
2) Modifying the design — the system design was modified to that presented in this paper; the interrupt handling sporadic simply places the data in a buffer and a cyclic object removes the data at an even rate (calculated to ensure that all the sensor objects get adequately fresh data) and calls the relevant sensor objects.
3) Modifying the translation to Ada 83 — the interrupt handling was implemented as a call to an Ada procedure (thus representing an optimised protected object in Ada 9X).

Given these modifications, the analysis indicated that the one soft task, TELECOMMANDS, would not meet its deadline in the worst case. In practice, the task did meet its deadline because of pessimism in:

1) the analysis techniques — some of the objects have offsets specified relative to other objects; the equations we were using assumed all tasks had a critical instance when this is clearly not the case (we now have more sophisticated analysis which will handle task offsets[23]).

2) the kernel — in order to take into account the overheads introduced by the kernel, we had to make some assumptions in the analysis techniques; these, on closer inspection, were a little pessimistic.[14]

3) the WCET tool — we estimate that our worst case execution time tool[15] is between 5-15% pessimistic because the tool does not model the m68020 internal pipeline and because some hardware times are data dependent and the tool has to assume the worst case.

6. Conclusions

The goal of the project was to illustrate that hard real-time systems can be programmed in a multi-tasking Ada environment, and yet give the same guarantees as those offered by the cyclic executive approach. The following points should be emphasised:

1) The use of a multi-tasking design introduced flexibility into the design; for example when the early design was shown not to meet its deadline it was not necessary to redo complex cyclic schedules. Instead the design could be easily altered and the schedulability analysis re-done.

2) The use of deadline monotonic scheduling, together with offsets between processes allowed input and output jitter to be kept to a minimum.

3) It is extremely important to model accurately the performance properties of the real-time operating system kernel if the scheduling analysis is to be relied on.

Acknowledgement

The authors would like to thank Paco Gomez Molinero, Fernando Gonzalez-Barcia and Tullio Vardanega for their help during the course of the project.

APPENDIX 1: Mapping HRT-HOOD to Ada 83

A.1 Introduction

In this appendix we consider the systematic translation of HRT-HOOD designs to Ada 83. The structure of the mappings given is based on the structure given for HOOD.[3] Other mappings are possible.

It is inevitable that a restricted subset of Ada will be required if a tool is to be designed that analyses Ada code for its worst case execution times. This subset *excludes* the following features:

- recursive or mutually recursive subprogram calls
- unbounded loop constructs
- dynamic storage allocation
- unconstrained arrays or types containing unconstrained arrays

Although periodic threads are implemented using an Ada delay statement, the schedulability analysis cannot cope with arbitrary delays in thread execution. Consequently we *do not allow* the application programmer to use

- the delay statement.

A.2 HRT-HOOD Translation

A.2.1 The Approach

It is widely accepted that Ada 83 lacks sufficient expressive power for programming hard real-time systems. Although Ada 9X has addressed many of these limitations, it will be some years before real-time Ada 9X development environments become available. Consequently the project attempted to provide much of the Ada 9X real-time functionality by making simple modifications to the York compiler and its stand-alone run-time system. Our approach to hard real-time system design requires the following to be supported by the implementation language and environment.

a) A large range of priorities — in Ada 83 there was no minimum range of priorities that an implementation had to support; in Ada 9X a minimum range of 32 priority levels is required.

b) Asynchronous (data-oriented) communication with bounded blocking — Ada 9X has introduced the notion of a protected object which can be used to decouple interacting tasks; a protected object enables the data to be shared between tasks to be encapsulated and operations to be defined which have automatic mutual exclusion. For single processor systems the mutual exclusion can be implemented by allocating a "ceiling" priority to the protected object; all operations are then executed at this priority. There is no direct equivalent facility in Ada 83.

c) Synchronisation with a monotonically increasing clock — Ada 9X allows a task to "delay until" a time in the future, where the time can either be specified by the time-of-day clock or the monotonically increasing clock. The latter is required to give a more accurate representation of a periodic task. Ada 83 simply allows a task to issue a relative delay.

d) Interrupt handling via protected objects — Ada 9X allows a protected operation to be called directly by an interrupt. In Ada 83, interrupts are mapped to task entries.

Support for a Large Priority Range

For most run-time support systems, increasing the range of priorities to be supported is a relatively simple matter. Furthermore ordering entry queues and a priority driven select statement can easily be added. However if the application area is using only protected tasks for communication and synchronisation (see below) and not the generalised rendezvous primitives, then a priority select is not required, priority entry queues are only required if more than one task can be queued on a protected task entry.

Support for Asynchronous (data-oriented) Communication

Many Ada 83 compilers already support the notion of a "passive" task. Passive tasks usually control access to shared data or are used to provide fast interrupt handlers, and therefore do not require an independent thread of control. Passive tasks are typically indicated by a pragma and the compiler will generate specific calls to the run-time systems.

In Ada 83 protected object semantics can be implemented by a passive task. For example the following task implements a protected object which has two protected access operations: op1 and op2; op1 is only accepted if an appropriate guard is open (it therefore represents a protected entry).

```
task PROTECTED is
    pragma PRIORITY(CEILING);
    pragma PROTECTED; -- recognised by the compiler
    entry OP1(...);
    entry OP2(...);
end PROTECTED;

task body PROTECTED is
    -- no local variables
begin
    loop
        select
            when G1 =>
                accept OP1(...) do
                    ...;
                end OP1;
        or
            accept OP2(...) do
                ...;
            end OP2;
        or
            terminate;
        end select;
    end loop;
end PROTECTED;
```

The structure can be recognised by the compiler; note that no run-time inheritance protocols are required as the PROTECTED task can be assigned a priority greater than its callers. The semantics for releasing tasks blocked on an entry can be implemented by the run-time system according to the Ada 9X protected object semantics.

Support for Periodic Task Execution

It is possible to provide the following simple run-time package which provides access to a monotonic clock, and therefore the following routine can be defined.

```
with MONOTONIC; use MONOTONIC;
package DELAY_SUPPORT is

    procedure DELAY_UNTIL(T : TIME);

end DELAY_SUPPORT;
```

A periodic task can take the form:

```
declare
    NEXT : TIME;
    INTERVAL : constant DURATION := ...;
begin
    NEXT := CLOCK + INTERVAL;
    loop
        -- code to be executed
        DELAY_UNTIL(NEXT);
        NEXT := NEXT + INTERVAL;
    end loop;
end;
```

Support for Interrupt Handling

The Ada 9X approach to interrupt handling is to allow interrupts to call a procedure in a protected object. Given the implementation of protected tasks described above it is relatively simple to allow an address clause to be placed on an entry. However, in our case study we did not have an optimised form of a protected task and therefore interrupt handling was too slow. Consequently we also allowed an interrupt address clause to be associated with an Ada procedure.

A.2 The Mappings

In this section we illustrate the mappings of HRT-HOOD to Ada 83. We make certain simplifications for the purpose of presentation. See Burns and Wellings for a full description of the mapping.[10]

Ada 83 mapping for a PROTECTED terminal Object <Name>

Let such an object have the following:

PSER for an operation which requires mutual exclusion.
FPSER for an operation which requires mutual exclusion and has a functional
activation constraint.

The specification of the package giving the provided operations is: (we assume appropriate "with" clauses):

```
package <NAME> is

    task OBCS is
        pragma CEILING_PRIORITY(CEILING);
        pragma PROTECTED;
        entry PSER(<PARAMETER PART>);
        entry FPSER(<PARAMETER PART>);
    end OBCS;

    procedure PSER(<PARAMETER PART>) renames OBCS.PSER;
    procedure FPSER(<PARAMETER PART>) renames OBCS.FPSER;

end <NAME>;
```

The following body has the same semantics as the Ada 9X protected records.

```
with CPU_BUDGETING; use CPU_BUDGETING;
package body <NAME> is

    procedure OPCS_PSER(<PARAMETER PART>) is separate;
    procedure OPCS_FPSER(<PARAMETER PART>) is separate; -- not shown
    procedure OPCS_FPSER_FAC is separate; -- not shown

    task body OBCS is
        -- no local variables
    begin
        loop
            select
                when OPCS_FPSER_FAC =>
                    accept FPSER(<PARAMETER PART>) do
                        OPCS_FPSER(<PARAMETER PART>);
                    end FPSER;
            or
                accept PSER(<PARAMETER PART>) do
                        OPCS_PSER(<PARAMETER PART>);
                end PSER;
            or
                terminate;
            end select;
        end loop;
    end OBCS;

end <NAME>;

separate(<NAME>);
procedure OPCS_PSER(<PARAMETER PART>) is
begin
    <OPCS_CODE>;
end OPCS_PSER;
```

Ada 83 mapping for a CYCLIC terminal Object <Name>

Let such an object have no interface. The specification of the package

```
package <NAME> is
end <NAME>;
```

```
package body <NAME> is

    procedure OPCS_PERIODIC_CODE is separate;

    task body THREAD is
       T : MONOTONIC.TIME;
       PERIOD : DURATION;
    begin
        DELAY_UNTIL(GET_START_TIME+OFFSET); -- if the THREAD has an offset
                                            -- GET_START_TIME returns the
                                            -- program's start time
        T:= CLOCK + PERIOD;
        loop
            begin
                OPCS_PERIODIC_CODE;
                DELAY_UNTIL (T);
                T := T + PERIOD;
            end;
        end loop;
    end;

separate(<NAME>);
procedure OPCS_PERIODIC_CODE is
begin
    <OPCS_CODE>;
end OP_NAME;
```

Ada 83 mapping for a SPORADIC terminal Object <Name>

Let such an object have the following:

START for an asynchronous operation which invokes the sporadic thread.

The package specification is:

```
package <NAME> is

    task OBCS is
        pragma PRIORITY(CEILING);
        pragma PROTECTED;
        entry START(<PARAMETER PART>);
        entry WAIT_START(<PARAMETER PART>);
    end OBCS;

    procedure START(<PARAMETER PART>) renames OBCS.START;

end <NAME>;
```

The package body is:

```
package body <NAME> is

    procedure OPCS_START(<PARAMETER PART>) is separate; -- not shown

    task body OBCS is
        START_OPEN : BOOLEAN := FALSE;
        T : MONOTONIC.TIME;
    begin
        loop
            select
                when not START_OPEN =>
                    accept START (<PARAMETER PART>) do
                        -- save params
                        T := MONOTONIC.CLOCK;
                        START_OPEN := TRUE;
                    end START;
            or
                when START_OPEN =>
                    accept WAIT_START (<PARAMETER PART>) do
                        -- write params and T
                        START_OPEN := FALSE;
                    end WAIT_START;
            or
                terminate;
            end select;
        end loop;

    end OBCS;

    task body THREAD is
        MAT : DURATION; -- minimum inter-arrival time
    begin
        loop
            OBCS.WAIT_START(<PARAMETER PART>);
            -- parameters includes T
            OPCS_START(<PARAMETER PART>);
            DELAY_UNTIL (MAT + T);
        end loop;
    end;
```

APPENDIX 2: Detailed Results

In this appendix we give more details of the timing of the control system and the execution environment. Our intention is that the information should be complete enough for others to reproduce our results.

The Real-time Properties of Cyclic Objects

The following table summarises the characteristics of the periodic threads. In it WCET is the time taken by the thread to execute its code and to execute any protected object entries. Computation Time is the time the thread executes including the overheads of the execution environment. The blocking time is the time the thread is blocked by a lower priority thread executing in a protected object.

Object Name	READ_BUS_IP
Object Number	C1
Period	1.00000E+01
WCET	1.76386E+00
Critical Level	HARD
Deadline	1.00000E+01
Offset	0.00000E+00
Priority	23
Computation Time	2.46386E+00
Block Time	1.37371E+00
Scheduling Result	TRUE
Deadline Met	6.99194E+00
Object Name	REAL_TIME_CLOCK
Object Number	C2
Period	5.00000E+01
WCET	2.82484E-01
Critical Level	HARD
Deadline	9.00000E+00
Offset	0.00000E+00
Priority	26
Computation Time	7.54484E-01
Block Time	3.72000E-01
Scheduling Result	TRUE
Deadline Met	3.52636E+00
Object Name	COMMAND_ACTUATORS
Object Number	C3
Period	2.00000E+02
WCET	2.12646E+00
Critical Level	HARD
Deadline	1.40000E+01
Offset	5.00000E+01
Priority	20
Computation Time	3.73845E+00
Block Time	1.37371E+00
Scheduling Result	TRUE
Deadline Met	1.35223E+01
Object Name	REQUEST_WHEEL_SPEEDS
Object Number	C4
Period	2.00000E+02
WCET	1.42574E+00
Critical Level	HARD
Deadline	2.20000E+01
Offset	0.00000E+00
Priority	18
Computation Time	2.35374E+00
Block Time	1.37371E+00
Scheduling Result	TRUE
Deadline Met	1.82297E+01
Object Name	CONTROL_LAW
Object Number	C5
Period	2.00000E+02
WCET	5.28458E+01
Critical Level	HARD
Deadline	2.00000E+02
Offset	5.00000E+01
Priority	8
Computation Time	5.67378E+01
Block Time	1.38224E+00
Scheduling Result	TRUE
Deadline Met	1.83506E+02
Object Name	PROCESS_DSS_DATA
Object Number	C6
Period	1.00000E+03
WCET	5.15615E+00
Critical Level	HARD
Deadline	4.00000E+02
Offset	2.00000E+02
Priority	6
Computation Time	6.31215E+00
Block Time	1.38224E+00
Scheduling Result	TRUE
Deadline Met	1.98386E+02
Object Name	REQUEST_DSS_DATA
Object Number	C7
Period	2.00000E+02
WCET	1.42574E+00
Critical Level	HARD
Deadline	1.70000E+01
Offset	1.50000E+02
Priority	19
Computation Time	2.35374E+00
Block Time	1.37371E+00
Scheduling Result	TRUE
Deadline Met	1.58760E+01
Object Name	CALIBRATE_GYRO
Object Number	C8
Period	1.00000E+03
WCET	6.91404E+00
Critical Level	HARD
Deadline	9.00000E+02
Offset	2.00000E+02
Priority	5

Computation Time	8.98204E+00
Block Time	1.38224E+00
Scheduling Result	TRUE
Deadline Met	3.89492E+02
Object Name	PROCESS_IRES_DATA
Object Number	C9
Period	1.00000E+02
WCET	8.20642E+00
Critical Level	HARD
Deadline	5.00000E+01
Offset	5.00000E+01
Priority	14
Computation Time	9.81842E+00
Block Time	1.37371E+00
Scheduling Result	TRUE
Deadline Met	4.41384E+01
Object Name	REQUEST_IRES_DATA
Object Number	C10
Period	1.00000E+02
WCET	1.42574E+00
Critical Level	HARD
Deadline	2.40000E+01
Offset	0.00000E+00
Priority	17
Computation Time	2.35374E+00
Block Time	1.37371E+00
Scheduling Result	TRUE
Deadline Met	2.33753E+01

The Real-time Properties of Sporadic Objects

Object Name	TELEMETRY_RESPONSE
Object Number	S1
WCET	3.19298E+00
Critical Level	HARD
Deadline	3.00000E+01
Gap	6.25000E+01
Priority	15
Computation Time	5.36098E+00
Block Time	1.37371E+00
Scheduling Result	TRUE
Deadline Met	2.87363E+01
Object Name	TELECOMMANDS
Object Number	S2
WCET	2.50060E+00
Critical Level	SOFT
Deadline	1.87000E+02
Gap	1.87000E+02
Priority	4
Computation Time	4.44060E+00
Block Time	3.72000E-01
Scheduling Result	FALSE
Deadline Met	-
Object Name	READ_YAW_GYRO
Object Number	S3
WCET	4.07858E+00
Critical Level	HARD
Deadline	1.00000E+02
Gap	1.00000E+02
Priority	12
Computation Time	6.70258E+00
Block Time	1.38224E+00
Scheduling Result	TRUE
Deadline Met	5.58463E+01
Object Name	MESSAGES_HERE
Object Number	S4
WCET	1.34240E+00
Critical Level	INTERRUPT
Gap	5.00000E+01
Priority	62
Computation Time	1.45040E+00
Object Name	TM_HERE
Object Number	S5
WCET	9.91602E-02
Critical Level	INTERRUPT
Gap	6.25000E+01
Priority	62
Computation Time	2.07160E-01
Object Name	Z1_HERE
Object Number	S6
WCET	9.91602E-02
Critical Level	INTERRUPT
Gap	1.00000E+02
Priority	62
Computation Time	2.07160E-01
Object Name	TC_HERE
Object Number	S7
WCET	9.91602E-02
Critical Level	INTERRUPT
Gap	1.87000E+02
Priority	62
Computation Time	2.07160E-01

Thread/Protected Object Interaction

Object Name	SERIAL_BUS_OP
Object Number	P1
WCET	3.95441E-01
Ceiling	22
Used By	C3, C4, C7, C10, S1
Object Name	TELEMETRY_RESPONSE.OBCS
Object Number	P2
WCET	1.45171E-01
Ceiling	24
Used By	C1 S1
Object Name	ECHO_OR_ERROR
Object Number	P3
WCET	3.04499E-01
Ceiling	25
Used By	C3, C4, C5, C6, C7, C8, C9, C10 S1, S2, S3

Object Name	TM_DATA_STORE
Object Number	P4
WCET	1.37371E+00
Ceiling	24
Used By	C1, C5, C8, C9 S1, S2, S3
Object Name	TELECOMMANDS.OBCS
Object Number	P5
WCET	1.45171E-01
Ceiling	24
Used By	C1 S2
Object Name	BUS_IP_FIFO
Object Number	P6
WCET	0.06000E+00
Ceiling	63
Used By	C1 S5
Object Name	SERIAL_BUS_IP
Object Number	P7
WCET	5.80204E-01
Ceiling	24
Used By	C1, C5, C6, C9
Object Name	INITIALISATION
Object Number	P8
WCET	6.34193E+00
Ceiling	27
Used By	C2, C3, C4, C5, C6, C7, C8, C9, C10
Object Name	EQUIPMENT_STATUS
Object Number	P9
WCET	1.34853E-01
Ceiling	11
Used By	C5 S2
Object Name	THRUSTERS
Object Number	P10
WCET	7.84509E-01
Ceiling	21
Used By	C3, C5
Object Name	WHEEL_COMMAND
Object Number	P11
WCET	7.63872E-01
Ceiling	21
Used By	C3, C5
Object Name	CONTROL_DUMPING
Object Number	P12
WCET	1.91515E+01
Ceiling	10
Used By	C5
Object Name	PROCESS_DEMAND
Object Number	P13
WCET	4.40894E+01
Ceiling	9
Used By	C5
Object Name	ATTITUDE
Object Number	P14
WCET	5.17816E-01
Ceiling	16
Used By	C5, C8, C9 S1, S2, S3
Object Name	DSS_ANGLE
Object Number	P15
WCET	1.55669E-01
Ceiling	7
Used By	C6, C8
Object Name	GYRO_STATE
Object Number	P16
WCET	1.38224E+00
Ceiling	13
Used By	C8 S2, S3
Object Name	READ_YAW_GYRO.OBCS
Object Number	P17
WCET	1.45051E-01
Ceiling	24
Used By	C1 S3

Protected Object Interactions

The following table summarises which protected objects make use of other protected objects. Note that to implement sporadic objects requires a synchronisation agent. This is in fact a form of protected object. The name given to one of these objects is "sporadic name_OBCS".

Object Name	SERIAL_BUS_OP
Used By Protected	P10, P11
Object Name	ECHO_OR_ERROR
Used By Protected	P4
Object Name	TM_DATA_STORE
Used By Protected	P12,13
Object Name	SERIAL_BUS_IP
Used By Protected	P13
Object Name	EQUIPMENT_STATUS
Used By Protected	P12, P13
Object Name	THRUSTERS
Used By Protected	P12
Object Name	WHEEL_COMMAND
Used By Protected	P13
Object Name	CONTROL_DUMPING
Used By Protected	P13

Execution Environment

The execution environment has certain characteristics which must be accounted for, if the analysis is to be accurate. The following table summarises the real-time characteristics of our execution environment. The entries are:

- INTERRUPT_CONTEXT_SWITCH_TIME

The time cost of a interrupt sporadic context switch.

- CONTEXT_SWITCH_TIME

  The time cost of a normal context switch.
- RELEASE_QUEUE_TIME

  The time cost of releasing a thread from the delay queue and moving it to the run queue (dispatch queue).
- DELAY_QUEUE_TIME

  The time cost of putting a thread in the delay queue.
- RELEASE_QUEUE_BLOCKING_TIME

  The blocking time cost of releasing a cyclic thread from the delay queue and moving it to the run queue.
- PROTECTED_RECORD_ENTER_TIME The time cost of entering a protected object.
- PROTECTED_RECORD_LEAVE_TIME

  The time cost of leaving a protected object.
- DISABLE_INTERRUPT_TIME

  The time cost of disabling interrupts.
- ENABLE_INTERRUPT_TIME

  The time cost of enabling interrupts.
- DELAY_EXPIRATION_TIME The maximum time between a delay expiring and the theoretical time at which it should expire (release jitter).
- MAX_NON_PREEMPTION_TIME

  The maximum period of non pre-emption exhibited by the execution environment.
- MAX_RUN_TIME_SYSTEM_OVERHEAD

  Worst case system overhead time.
- MAX_FREQUENCY_RTS_OVERHEAD Period of MAX_RUN_TIME_RTS_OVERHEAD
- PRIORITY_FIRST

  Lowest software priority allowed for this hardware platform.
- PRIORITY_LAST

  Highest software priority allowed for this hardware platform.
- BLOCKING_APPROACH

  Either IPCI or DISABLE_INTERRUPTS

Interrupt Context Switch Time	5.40000E-02
Context Switch Time	1.64000E-01
Release Queue Time	6.00000E-02
Delay Queue Time	8.40000E-02
Release Queue Blocking Time	0.00000E+00
Protected Record Enter Time	8.80000E-02
Protected Record Leave Time	1.40000E-01
Disable Interrupt Time	4.00000E-03
Enable Interrupt Time	4.00000E-03
Delay Expiration Time	0.00000E+00
Max Non Preemption Time	3.72000E-01
Maximum Run Time System Overhead	3.28000E-01
Max Frequency RTS Overhead	1.00000E+01
Priority First	2
Priority Last	63
Blocking Approach	IPCI

References

1. *York Ada Compiler Environment (York ACE) Reference Guide*, York Software Engineering Limited (1991). (Release 5.1)
2. European Space Agency, ''HOOD Reference Manual Issue 3.0'', WME/89-173/JB (September 1989).
3. European Space Agency, ''HOOD Reference Manual Issue 3.0'', WME/89-173/JB (September 1989).
4. N. Audsley, A. Burns, M. Richardson, K. Tindell and A. Wellings, ''Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling'', *Submitted to Software Engineering Journal* (to appear).
5. C. Bailey, ''Survey of Typical Space Applications'', Task 6 Deliverable on ESTEC Contract 9198/90/NL/SF, British Aerospace Space Systems Ltd. (September 1991).
6. C. Bailey, ''Software Requirements Document for the Olympus AOCS'', Task 10 Deliverable on ESTEC Contract 9198/90/NL/SF, British Aerospace Space Systems Ltd. (March 1992).
7. C.M. Bailey, A. Burns, E. Fyfe, F. Gomez-Molinero and A.J. Wellings, ''Implementing Hard Real-time Systems: A Case Study'', *Proceeeding International Symposium on Real-time Embedded Processing for Space Applications, Les Saintes-Maries-de-la-Mer, France* (November 1992).
8. A. Burns and A.J. Wellings, ''Real-time Ada: Outstanding Problem Areas'', *Proceedings of the 3nd International Workshop on Real Time Ada Issues, ACM Ada Letters, Ada Letters* **X**(4), pp. 5-14 (1990).
9. A. Burns and A.J. Wellings, ''Usability of the Ada Tasking Model'', *Proceedings of the 3nd International Workshop on Real Time Ada Issues, ACM Ada Letters, Ada Letters* **X**(4), pp. 49-56 (1990).
10. A. Burns and A.J. Wellings, ''Development of a Design Methodology'', Task 3 Deliverable on ESTEC Contract 9198/90/NL/SF, Department of Computer Science, University of York (September 1991).
11. A. Burns and A.J. Wellings, ''Definition of Tools'', Task 4 Deliverable on ESTEC Contract 9198/90/NL/SF, Department of Computer Science, University of York (September 1991).
12. A. Burns and A.J. Wellings, ''Designing Hard Real-time Systems'', pp. 116-127 in *Ada: Moving Towards 2000, Proceeedings of the 11th Ada-Europe Conference, Lecture Notes in Computer Science Vol 603*, Springer-Verlag (1992).
13. A. Burns and A.J. Wellings, *Hard Real-time HOOD: A Design Method for Hard Real-time Ada 9X Systems*, Towards Ada 9X, Proceedings of 1991 Ada UK International Conference, IOS Press (1992).
14. A. Burns, A.J. Wellings and A.D. Hutcheon, ''The Impact of an Ada Run-time System's Performance Characteristics on Scheduling Models'', in *Ada sans frontieres Proceeedings of the 12th Ada-Europe Conference, Lecture Notes in Computer Science*, Springer-Verlag (to appear).
15. C.H. Forsyth, ''Implementation of the Worst-Case Execution Time Analysier'', Task 8 Volume E, Deliverable on ESTEC Contract 9198/90/NL/SF, York Software Engineering Limited, University of York (June 1992).
16. Intermetrics, ''Draft Ada 9X Mapping Document, Volume II, Mapping Specification'', Ada 9X Project Report (December 1991).
17. J.Y.T. Leung and J. Whitehead, ''On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks'', *Performance Evaluation (Netherlands)* **2**(4), pp. 237-250 (December 1982).
18. C.L. Liu and J.W. Layland, ''Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment'', *JACM* **20**(1), pp. 46-61 (1973).
19. C.D. Locke, ''Software architecture for hard real-time applications: cyclic executives vs. fixed

priority executives'', *Real-Time Systems* **4**(1), pp. 37-53, Real-Time Syst. (Netherlands) (March 1992).

20. M. Pilling, A. Burns and K. Raymond, ''Formal Specification and Proofs of Inheritance Protocols for Real-Time Scheduling'', *Software Engineering Journal (to appear)* (1990).

21. L. Sha, R. Rajkumar and J. P. Lehoczky, ''Priority Inheritance Protocols: An Approach to Real-Time Synchronisation'', *IEEE Transactions on Computers* **39**(9), pp. 1175-1185 (September 1990).

22. L. Sha and J. B. Goodenough, ''Real-Time Scheduling Theory and Ada'', *IEEE Computer* (April 1990).

23. K. Tindell, ''Using Ofset Information to Analyse Static Pre-emptive Scheduled Task Sets'', YCS 182, Department of Computer Science, University of York (September 1992).