Tutorial for the Open-Source DREAM Tool *

Draft v1.2

Gabor Madl, Nikil Dutt

{gabe, dutt}@ics.uci.edu

Center for Embedded Computer Systems,
University of California, Irvine CA 92697

Abstract

This paper describes practical applications of the *Distributed Real-time Embedded Analysis Method* (DREAM) open-source tool for the simulation, optimization, and formal verification of distributed real-time embedded (DRE) systems. We present several detailed case studies and describe how we utilize the UPPAAL model checker and the Verimag IF toolset for the real-time analysis of these systems.

1 Introduction

DREAM is a model-based analysis tool which focuses on system design at higher abstraction levels than methods which build on source code analysis. The design principle for DREAM is to construct a tool that can be practically used for *component-based* system development. This design principle is motivated by software engineering practices that address the complexity of system design by introducing patterns and methods for *code reuse*.

This paper describes the DREAM reusable verification framework which aims to support this code reuse process using formal methods. The discussion is based on the DREAM 0.5 ALPHA version, and the capabilities of DREAM will likely be extended in future versions. DREAM focuses on component behaviors that are easy to observe and measure, such as the best case

*DREAM is available for download at `http://dre.sourceforge.net`.

and worst case time of the required computation, priorities between tasks, and the required energy to carry out a specific task on a specific platform. DREAM captures tasks at a rather high-level abstraction, a task is assumed to be idle, waiting to be scheduled, executing, or possibly preempted. Dependencies between tasks are explicitly captured in DREAM.

DREAM models DRE systems as distributed multi-threaded event-driven real-time systems using fixed-priority scheduling. DREAM implements a discrete event simulator with clear execution semantics. Simulations are used for random testing and in the case of non-preemptive scheduling for exhaustive verification as well. DREAM is capable of automatically generating a formal timed automata representation from the internal discrete event model providing a way for the automatic verification of real-time properties [1]. DREAM also provides a simulation-based exhaustive verification method for real-time systems using non-preemptive fixed-priority scheduling and a design-time method for power management policy synthesis using timed automata model checking techniques. DREAM also implements a priority optimization method based on genetic algorithms.

Domain-specific modeling languages (DSMLs) are languages created for a specific, well-defined application domain. This approach is rather different from mainstream modeling efforts that focus on specifying a large and generic modeling language to be used in a wide range of applications, such as *UML*. DSMLs in our approach are defined using *meta-modeling* [2] therefore the designer has the option of defining languages that have well defined semantics and are a good fit for a problem domain. Large-scale systems that involve several application domains are modeled as a *composition* of DSMLs. We believe that defining semantics to smaller modeling languages and their composition is more likely to succeed than to define it for a large generic modeling language.

The DREAM tool builds on the *Analysis Language for Distributed, Embedded and Real-time Systems* (ALDERIS) DSML. ALDERIS is a specification language for power aware distributed real-time embedded systems. The language captures dense real-time properties on a distributed platform, shutdown-based energy savings methods as well as frequency- and voltage-scaling. ALDERIS provides a way for the design-time formal verification of system models as well as automated simulation. ALDERIS has both a visual and textual XML syntax allowing designers to choose the representation that they prefer.

Section 2 describes the motivation for creating the DREAM tool, Section 3 explains the principles used to design DREAM, Section 4 describes the mod-

eling language for DREAM, Section 5 introduces some models used as case studies, Section 6 describes the timed automata-based verification methods, Section 7 discusses the simulation-based real-time verification method for non-preemptive systems, Section 8 reviews tools aimed at the same problem domain, and Section 9 presents concluding remarks. The Appendix gives examples on how DREAM models can be specified using the XML-based DRE architecture description language (ADL) for distributed real-time embedded systems.

2 Motivation

Component-based middleware [3, 4, 5] is an emerging platform supporting key functional and quality of service (QoS) needs of distributed real-time embedded (DRE) systems. Components in the software context refer to reusable pieces of code, which can be configured and composed together to provide a service. Component-based software development shifts the focus from writing new code to the composition of existing and tested components. In the software context components do not refer to actual hardware elements, and do not necessarily have their own threads of execution. Generic purpose modeling languages tailored towards component-based software development (such as *UML 2.0*) often overlook this detail and do not define a clear method for the mapping of components to the hardware architecture.

Verification has a richer history and wider acceptance in hardware design where correct-by-construction engineering is an economic necessity. The industrial application of formal methods have been estabilished in the range of small systems (*e.g. SoCs*) and electronic system-level (ESL) design. The term *verification* in the hardware design context is often used to describe the rigorous testing process supporting hardware design and often does not refer to *formal verification*. The increasing complexity has shifted the focus of design methodologies from RTL-level analysis and languages (such as *VHDL* and *Verilog*) towards transaction-level analysis and *SystemC*. Commercial tools supported by design automation companies are dominantly based on testing and simulations, with only a few exceptions (such as *SMV* by Cadence or *SLAM* by Microsoft among others). Simulations, however, can cover only a small part of the design space therefore they do not substitute, rather complement formal verification. Real-life production SoC systems are usually orders of magnitude more complex than what can be efficiently verified and the gap between production and verification capability seems to increase exponentially. Verification is a time- and money-consuming pro-

cess which is a major barrier in the design of complex multi-processor SoC systems.

3 Design and Implementation

The driving force behind the design of the DREAM tool is to create a tool which captures high-level system design in a formal setting. As discussed in the Introduction, DREAM focuses on component behaviors that are easy to observe and measure, such as real-time properties and power consumption. Tasks are also captured at a rather high level abstraction, using only 4 states: (idle, wait, run, and preempted). The main reason for the agressive abstraction is to allow the real-time verification of a large number of tasks on heterogeneous platforms. Task computations are omitted for the timed automata-based real-time verification, but can be captured during simulations (including the exhaustive simulation-based verification). Capturing task computations is implemented as a semi-automatic feature and is explained in Section 7.

Computations are driven by timers which publish events at a given rate. In this paper we assume that all the timers are synchronized. This assumption can often be relaxed in practice. For example, the method is applicable if the time drifts between the timers can be bounded. We can compensate for this uncertainty by specifying larger execution intervals for individual tasks.

Some implementations of DRE systems do not directly correspond to this model of computation by allowing arbitrary drifts between the timers. Although DREAM can theoretically capture this model of computation several practical problems arise. First, the drifting timers will surely blow up the state space during verification as all combinations of all timers have to be considered by the model checker. Second, and more importantly, this step turns the system unpredictable, as it is no longer possible to estimate which tasks can be preempted by other tasks (or enabled at the same time in the non-preemptive case). Allowing drifting timers requires the designer to artificially increase the deadlines to compensate for *any* higher priority task that might preempt lower priority tasks. Using non-preemptive scheduling does not overcome this problem as same-priority tasks may receive events from remote tasks – which in the case of arbitrarily drifting clocks means that they might become enabled at any time – and the designer has to compensate for tasks that might receive an event from a remote task by increasing the deadlines.

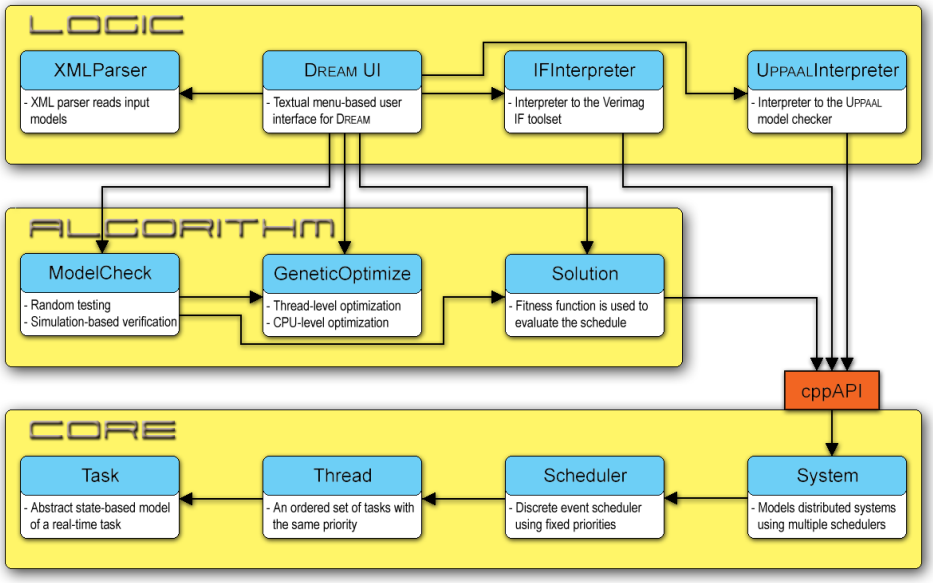

Figure 1: The Modular DREAM Design

A totally asynchronous system also has advantages as it is easier to be implemented since no synchronizations have to be maintained. DREAM provides a random testing service which is applicable to asynchronous unconstrained systems. This development model, however, is not applicable to hard real-time systems because formal guarantees for real-time properties can hardly be obtained in practice. DREAM provides a random testing service that might be useful to detect bugs in totally asynchronous systems, but verification methods do not scale well for these types of systems.

The design of DREAM is split up into three major modules; `Core` implements the discrete event scheduler on a preemptive distributed platform, `Algorithm` implements various algorithms for verification and optimization that build on the `Core`, and `Logic` implements the DREAM UI and the interpreters which create timed automata models from the internal representation in the `Core`.

The `Core` implements systems with varying complexity. The `Task` class implements a real-time task which models computations in the system. Tasks may have best and worst case execution times, and sub-priorities for non-preemptive scheduling. Tasks can be assigned to threads, which are a list of tasks with a priority for preemptive scheduling. A task's priority

is the priority of the thread – tasks that are assigned to the same thread have the same priorities. A processing node is represented as a fixed priority scheduler. A scheduler manages a thread-pool of (possibly) several threads (with distinct priorities). Higher priority threads are favored against lower priority threads, whenever a high priority thread becomes enabled it preempts any lower priority threads that are running. DREAM can simulate hyper-threaded systems as well where several threads may run concurrently on the same execution node at the same time. The `System` class represents a distributed system consisting of several processing nodes and provides a C++ API for the other two modules. The C++ API uses the visitor pattern to get access to the `Core`.

The `Algorithm` module uses this C++ API to obtain direct pointers to the tasks using the visitor pattern. Using the pointers the algorithm can optimize parameters by running several simulations, and specify whether best, worst, or random execution times are assumed for each task. The `Solution` class maintains the connection to the `Core` using the pointers and it can compute a fitness function for the current system model. If the system is schedulable 0 is returned, otherwise an error value is computed from the number of tasks that have missed their deadlines. The `GeneticOptimize` class uses genetic algorithms to optimize the scheduling by generating several possible priority assignments (using the `Solution` class). 2-parent and 1-parent mutations are used on the best solutions to obtain even better ones, while the worst candidates are replaced by random solutions. The `ModelCheck` class also builds on the `Solution` class. It implements simulation-based model checking for systems using non-preemptive scheduling. The method builds on the `Core` which updates the states and clocks for the tasks for each event and checks whether a task's execution time is longer than its deadline.

The `Logic` module implements the text-based menu from which DREAM services can be accessed. The UPPAAL and IF interpreters generate timed automata models directly from the `Core`. An XML parser is used to read models into DREAM.

DREAM is implemented in ANSI C++. The source code compiles using gcc for Linux as well as Visual Studio 6.0 and 7.1 compilers, and should be easy to port to other platforms as well. STL datastructures are used within DREAM for the sake of simplicity. The implementation is not particularly optimized but is relatively easy to read. All classes and methods are documented and available online at `http://dre.sourceforge.net/doc`. Supported build environments include GNU make and automake, KDevelop, Eclipse, and Visual Studio 6.0 and 7.1 (.NET). A design principle was not

to use runtime type information (RTTI) to allow the use of DREAM in embedded systems as well.

4 The ALDERIS Domain-specific Modeling Language

This section describes the syntax and the formal semantics of the ALDERIS language. We formalize an abstract model of computation that can express dense real-time properties and power consumption in a common semantic domain. We propose a platform-based analysis of DRE systems consisting of two major aspects: *dependency*, which describes various relations and dependencies between tasks, and *platform*, which specifies the platform that executes the tasks. We capture both these aspects in ALDERIS by specifying the event flow between tasks and their mappings to platform processors.

4.1 Syntax

The ALDERIS model of computation is a tuple $M = \{T, C, TR, PR\}$ where:

- T is a set of *tasks*,
- C is a set of *event channels*,
- TR is a set of *timers*, which are special tasks that publish events at a given rate,
- PR is a set of *platform processors*.

Tasks and timers are assigned to execute on a specific processor. The processor associated with a given task or timer is specified by the map $\mathsf{Processor} : T \cup TR \rightarrow PR$. Timers generate periodic events as specified by the map $\mathsf{Period} : TR \rightarrow \mathbb{N}^+$. Tasks are attributed by the following properties:

- $\mathsf{p}(t)$ is the priority of the task specified by the mapping $\mathsf{p} : T \rightarrow \mathbb{N}^+$,
- $\mathsf{sp}(t)$ is the sub-priority of the task specified by the mapping $\mathsf{sp} : T \rightarrow \mathbb{N}^+$,
- $\mathsf{dl}(t)$ is the deadline of the task specified by the mapping $\mathsf{dl} : T \rightarrow \mathbb{N}^+$,
- $\mathsf{wcet}(t)$ is the worst case execution time of the task specified by the mapping $\mathsf{wcet} : T \rightarrow \mathbb{N}^+$,

- $\mathsf{bcet}(t)$ is the best case execution time of the task specified by the mapping $\mathsf{bcet} : T \rightarrow \mathbb{N}^+$,

We write $\mathsf{State}(t, x)$ to denote the state of t at (global) time x_g: $(\forall t \in T)(\forall x \in \mathbb{N})\ \mathsf{State}(t, x) \in \{\mathtt{init}, \mathtt{wait}, \mathtt{run}, \mathtt{pass}, \mathtt{error}\}$

4.1.1 System State

The state of the system is defined as the composite state of its tasks and timers. For a dynamic element a, we write $\mathsf{State}(a, x)$ for the value of a at (global) time x. We now define the state domain of each model elements.

- $(\forall t \in T)(\forall x \in \mathbb{R}^+)\ \mathsf{State}(t, x) \in \{\mathtt{init}, \mathtt{wait}, \mathtt{run}, \mathtt{pass}\}$
- $(\forall r \in TR)(\forall x \in \mathbb{R}^+)\ \mathsf{State}(r, x) = \mathrm{mod}(x, \mathsf{Period}(r))$, where the function $\mathrm{mod}(x, y)$ returns the remainder of dividing x by y.

For a given computation model $M = \{T, C, TR, PR\}$, the state of the overall DRE model is given by:

$$\mathsf{State}(M, x) = \prod_{a \in T \cup TR} \mathsf{State}(a, x)$$

Note that the state of every system element is well-defined at any time x and so does the composite state of the system.

4.2 Time and Clocks

Time in the ALDERIS model of computation is continuous, system clocks (global and local) are assigned a value in $\mathbb{R}$, the set of real numbers. The current global clock value is denoted by x_g. We associate with each timer $tr \in TR$ a local clock c_{tr} that indicates the time remaining in the current cycle. We assume that timers are all triggered at the start of execution $x_g = 0$. We associate with each task t a local clock c_d that progresses linearly and is used to check whether the deadline is met. We also introduce for each task t a clock c_e that indicates the time spent in the execution state in the current execution cycle. Time in the model progresses linearly: $\frac{dx_g(t)}{dt} = 1, (\forall t \in T)(\frac{dc_d(t)}{dt} = 1)$, however we allow the value of local clocks to be reset to 0.

We associate a finite number of QoS-levels with each task. A QoS-level is a set of properties, a 2-tuple $Q = \{\mathsf{sf}, \mathsf{pw}\}$ where $\mathsf{sf}(t)$ is the slow factor that specifies the rate of the local clock relative to the global clock specified

by the mapping $\mathsf{sf} : T \rightarrow \mathbb{N}^+$ and $\mathsf{pw}(t)$ is the energy consumption of the task over a time unit specified by the mapping $\mathsf{pw} : T \rightarrow \mathbb{N}^+$. We do not define the time unit in seconds as it is irrelevant, the user can define any time unit. The slow factor sf determines how fast the task is executing by specifying the speed of the clock $\frac{dc_e(t)}{dt} = \frac{1}{\mathsf{sf}}$.

4.3 Semantics

We assume a fixed-priority scheduler which schedules tasks preemptively based on their priorities. Each scheduler is associated with a platform processor $pr \in PR$. We assume that at most one task can execute at a platform processor at any time. In addition, tasks depend on triggering tasks or timers. Tasks are executed as soon as they are enabled by system events and they are the highest priority task enabled. Real-time embedded systems can be modeled by the composition of timers, dynamic computation tasks, and priority assignments. We assume that communications through shared variables or common coupling are not considered in the model.

4.4 Specifying the ALDERIS DSML using Meta-modeling

The ALDERIS language is expressive enough to capture a wide range of DRE systems, as we will show in this paper. This section demonstrates how the concepts of *model-integrated computing (MIC)* [2] can be utilized to define the ALDERIS DSML. MIC promotes a metamodel-based approach for powerful domain-specific abstractions that capture key concepts and concerns of DRE systems, such as their structure, behavior, and environment, as well as the QoS properties they must satisfy. Experience to date has shown that models are essential throughout the DRE system lifecycle, including the design, configuration, integration, and analysis phases. MIC adopts the four-layered metamodeling architecture of the Model Driven Architecture (MDA) that has been standardized by the Object Management Group (OMG). MIC can be viewed as an enhancement of MDA that is tailored towards system design via domain-specific modeling languages. This approach provides a practical visual language that can be used by DRE domain experts that are not necessarily familiar with formal methods.

The *Generic Modeling Environment* (GME) [6] is an MIC toolsuite that provides a visual interface to simplify the development of domain-specific modeling languages (DSMLs). GME contains a metamodeling environment that supports the definition of paradigms, which are type systems that describe the roles and relationships in particular domains. GME has

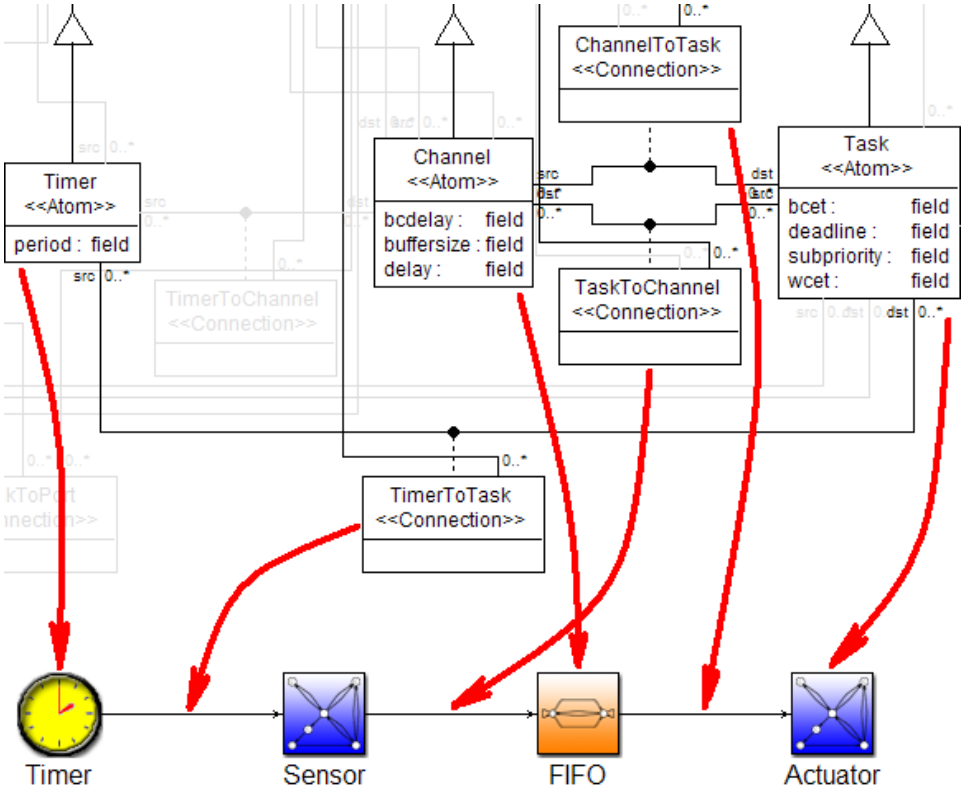

Figure 2: Specifying the ALDERIS DSML using Meta-modeling

a flexible object-oriented type system that supports inheritance and instantiation of elements of DSMLs. GME allows to specify a modeling language using meta-modeling. GME is not part of the DREAM package because of copyright issues but can be freely downloaded at `http://www.isis.vanderbilt.edu/projects/gme`.

Figure 2 illustrates the specification of the ALDERIS language using the GME meta-model, which is a variation of UML class diagrams. The figure shows a part of the ALDERIS meta-model with its corresponding visual representation in GME. The curvy arrows show how individual modeling elements and their relations are defined by different parts of the meta-model. The ALDERIS modeling language is automatically synthesized from the meta-model by the GME tool. The next section describes how the synthesized ALDERIS DSML is used to model power aware DRE systems.

5 Example DRE application case studies

In this section we describe several case studies that serve as examples and "test-benches" to evaluate various optimization and analysis methods. We use the visual ALDERIS language to show the applications. Please see the Appendix for the textual representation of the same models. The textual data is directly used to drive the DREAM tool.

5.1 Single CPU Non-preemptive Real-time CORBA Application

The first example shown is a single-processor non-preemptive Real-time CORBA application shown in Figure 3 and described using the DREAM XML specification in Subsection 10.1. The application consists of 8 components, namely `GPS`, `INS`, `ADC`, `RADAR1`, `RADAR2`, `AIRFRAME`, `NAV_DISPLAY`, and `TACTICAL_STEERING`. The visual modeling language used for building the applications have been created using the Generic Modeling Environment (GME).

The publisher/subscriber communication pattern [7] is used in the example. In this model the publisher notifies all the subscribers when it has data available for them. In this model the subscribers do not need to contin-

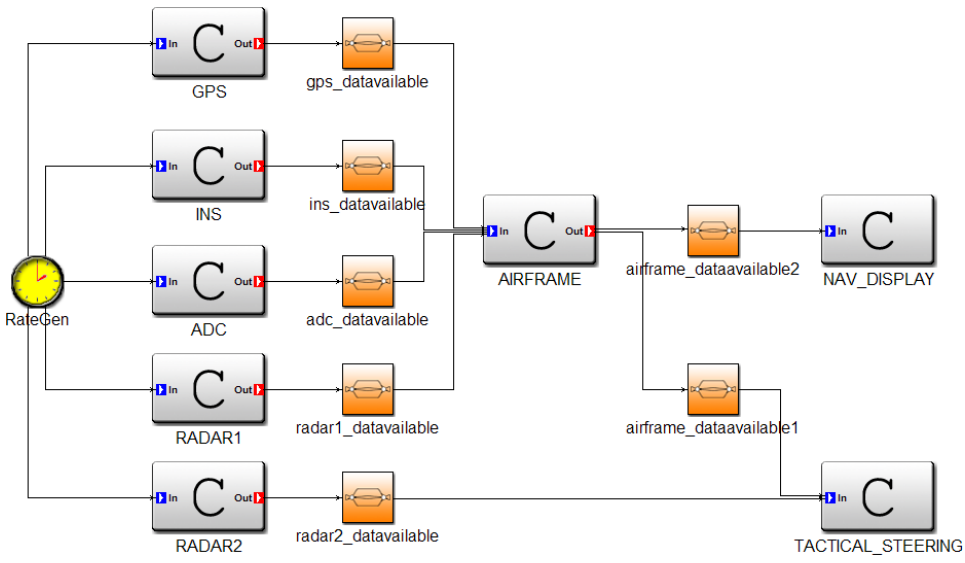

Figure 3: Single CPU Non-preemptive Real-time CORBA Application

Task	WCET	BCET	DL	SP
GPS	38	21	168	2
INS	45	37	621	6
ADC	31	23	825	7
RADAR1	62	48	370	4
RADAR2	52	32	53	0
AIRFRAME	63	54	64	3
NAV_DISPLAY	33	21	313	5
TACTICAL_STEERING	77	65	78	1

Channel	WCDelay	BCDelay	BufferSize
gps_dataavailable	0	0	2
ins_dataavailable	0	0	2
adc_dataavailable	0	0	2
radar1_dataavailable	0	0	2
radar2_dataavailable	0	0	2
airframe_dataavailable1	0	0	2
airframe_dataavailable2	0	0	2

Table 1: Timing Information for the Single CPU Non-preemptive Example

uously poll the publisher therefore the utilization of the network is better. However, it might be harder to detect failures or lossy channels using this communication pattern.

In the single CPU non-preemptive example all components are deployed on the same thread therefore a simple non-preemptive scheduling is used between the components. The computation is driven by a timer (RateGen) which publishes events with a predefined rate – 1 Hz – therefore it will publish an event in every second. The GPS, INS, ADC, RADAR1, and RADAR2 components are notified at the same time therefore they become enabled at the same time. For the sake of simplicity all the components contain only one task.

Once a task finishes its execution it pushes an event to the AIRFRAME – in the case of RADAR2 to TACTICAL_STEERING – through an event channel. The event channel is built on the asynchronous method invocation (AMI) [8] feature of the CORBA specification. The event channel is the implementation of an agent that manages the event passing between tasks. The publisher task pushes the event to the event channel and resumes its execution, rather than waiting for the subscriber to process the event. When

the subscriber is ready to process the event it notifies the event channel, which issues the remote method on the subscriber as the publisher's agent. The event channel also buffers events alleviating the synchronization needs for the communication.

The CORBA specification also specifies a standard for remote method calls. In contrast, the example shown in Figure 3 does not model any remote method calls. Capturing remote method calls in asynchronous event-driven systems is an easy way to blow up the state space. However, we can use software engineering to overcome this problem. In most real-time CORBA implementations remotely initiated actions have their dedicated thread. In the examples that follow we assume that the thread is always ready to serve requests and is not blocking. Although this assumption is overly optimistic in some cases, we can always add more ORB threads or let the ORB's Portable Object Adapter (POA) manage dynamic number of threads in the thread pool. The threading model described above provides a way to agressively abstract out remote method calls from the model. For local method calls we simply add the WCET of the called and caller tasks. For remote method calls we also add the worst-case delay of the channel to the WCET of the call chain.

5.2 Multiple CPU Non-preemptive Real-time CORBA Application

In the multiple CPU non-preemptive example we use the same dependencies shown in Figure 3, but the components are deployed on a 3-processor platform. Each processor schedules tasks non-preemptively on a single processor. The mapping of tasks to threads and processors is shown in Figure 4 and described using the DREAM XML specification in Subsection 10.2.

Although the dependencies are the same the parallel execution of several tasks greatly improves the performance of the overall application. The decreasing task deadlines result in faster response times. Some deadlines, however, increase to compensate for priority inversions introduced by the asynchronous communication between the processors. For example, the execution order between the `NAV_DISPLAY` and `TACTICAL_STEERING` components is determined by the delays of the `airframe_dataavailable1` and

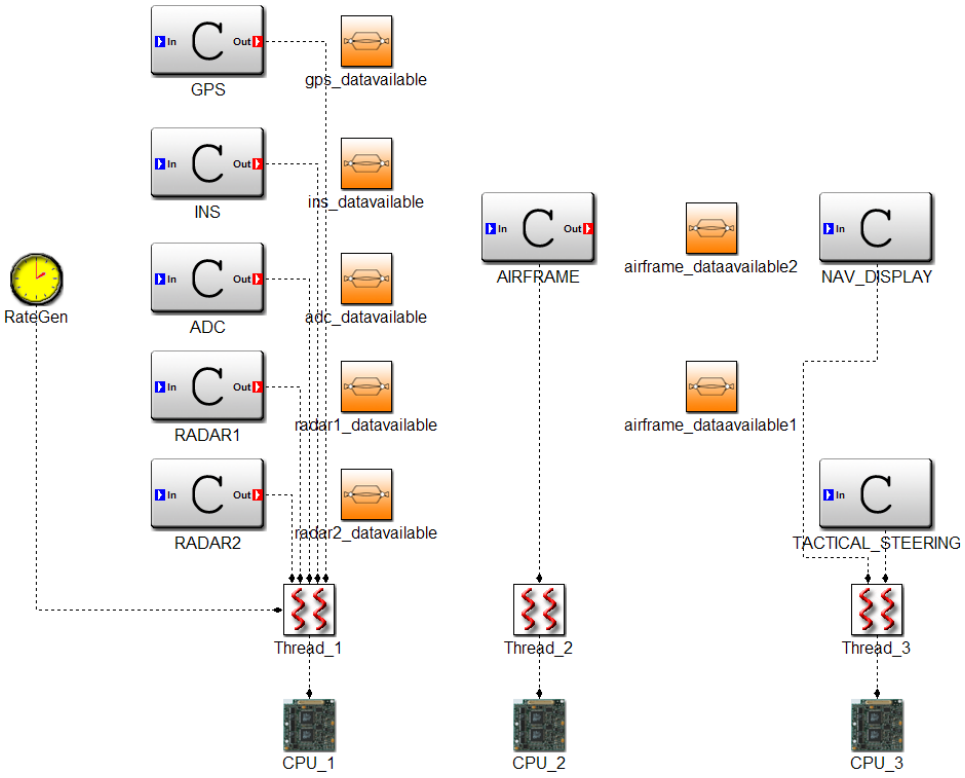

Figure 4: Multiple CPU Non-preemptive Real-time CORBA Application

Task	WCET	BCET	DL	SP
GPS	38	21	39	2
INS	45	37	146	6
ADC	31	23	177	7
RADAR1	62	48	101	4
RADAR2	52	32	53	0
AIRFRAME	63	54	64	3
NAV_DISPLAY	33	21	318	5
TACTICAL_STEERING	77	65	111	1

Channel	WCDelay	BCDelay	BufferSize
gps_dataavailable	5	3	2
ins_dataavailable	5	3	2
adc_dataavailable	3	1	2
radar1_dataavailable	2	0	2
radar2_dataavailable	2	1	2
airframe_dataavailable1	2	1	2
airframe_dataavailable2	2	1	2

Table 2: Timing Information for the Multiple CPU Non-preemptive Example

airframe_dataavailable2 channels; whichever becomes enabled first will be executed first as non-preemptive scheduling is used within a single thread. The single CPU non-preemptive example, in contrast, enforced a deterministic schedule as the NAV_DISPLAY and TACTICAL_STEERING components became enabled in the same clock cyle. This simple example highlights the need to explicitly capture dependencies and demonstrates the anomaly that priorities cannot necessarily enforce a deterministic schedule in real-life DRE systems. The parameters for the application are shown in Table 5.2.

5.3 Small Distributed Non-preemptive Real-time CORBA Application

The small distributed non-preemptive example is the first example discussed in this paper that uses several timers to drive the computations. The appli-

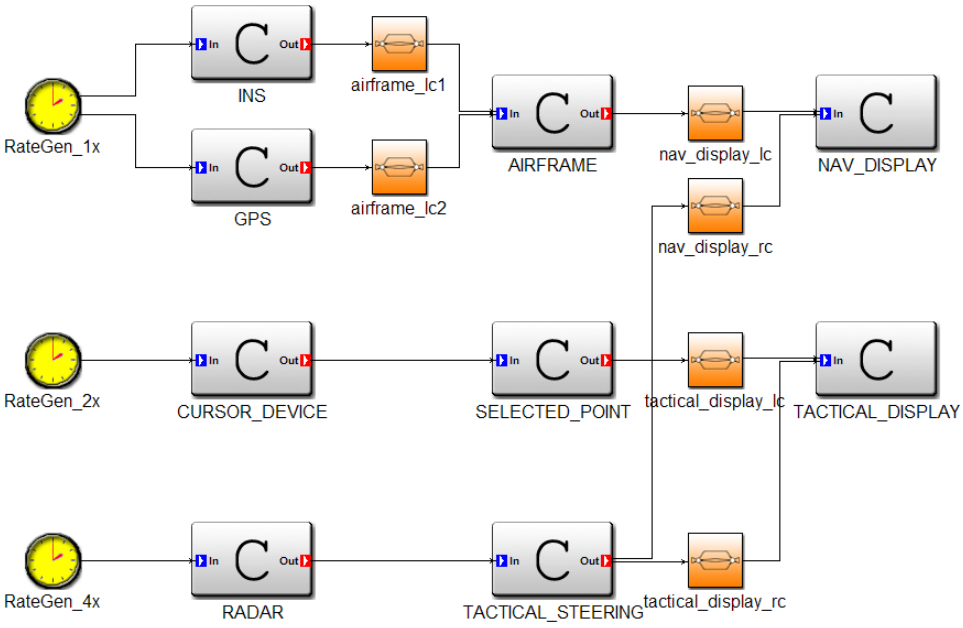

Figure 5: Small Distributed Non-preemptive Real-time CORBA Application

Task	WCET	BCET	DL	SP
INS	32	27	872	6
GPS	29	22	869	7
AIRFRAME	80	67	920	8
NAV_DISPLAY	19	13	859	9
CURSOR_DEVICE	18	13	155	3
SELECTED_POINT	24	17	161	4
TACTICAL_DISPLAY	21	9	158	5
RADAR	12	8	84	1
TACTICAL_STEERING	16	11	88	2

Table 3: Timing Information for the Small Distributed Non-preemptive Example

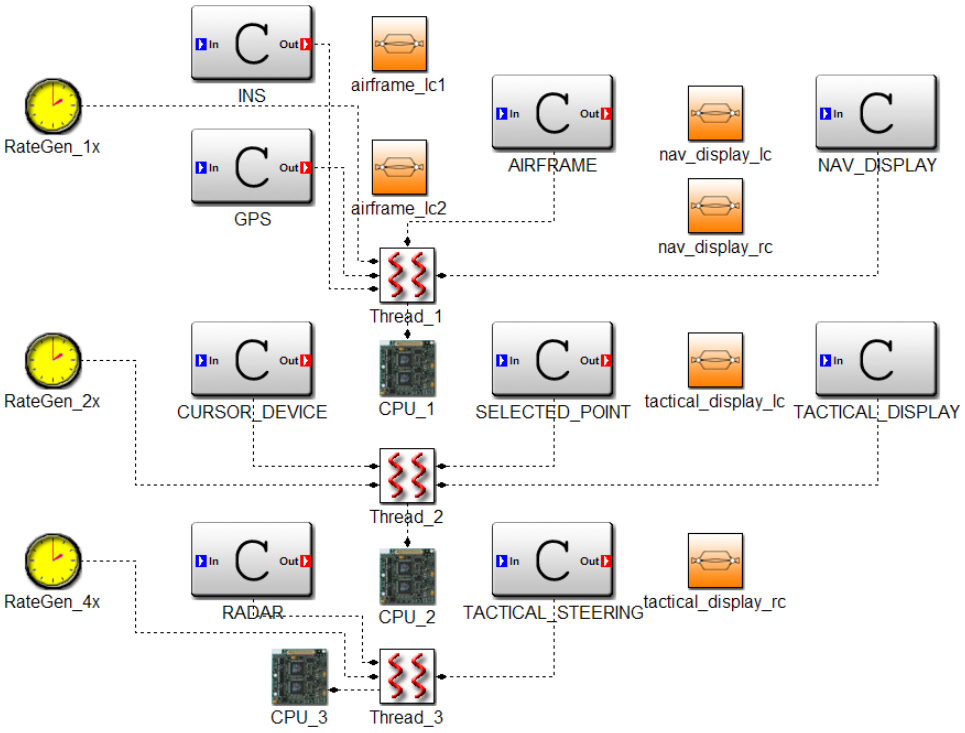

Figure 6: Small Distributed Non-preemptive Real-time CORBA Application

Channel	WCDelay	BCDelay	BufferSize
airframe_lc1	0	0	2
airframe_lc2	0	0	2
tactical_display_lc	0	0	2
nav_display_lc	0	0	3
tactical_display_rc	2	1	2
nav_display_rc	2	0	5

Table 4: Timing Information for the Small Distributed Non-preemptive Example

cation consists of 3 threads mapped onto separate processors.

5.4 Small Distributed Preemptive Real-time CORBA Application

The small distributed preemptive example is the first preemptive example discussed in this paper. The dependencies are the same as in the small

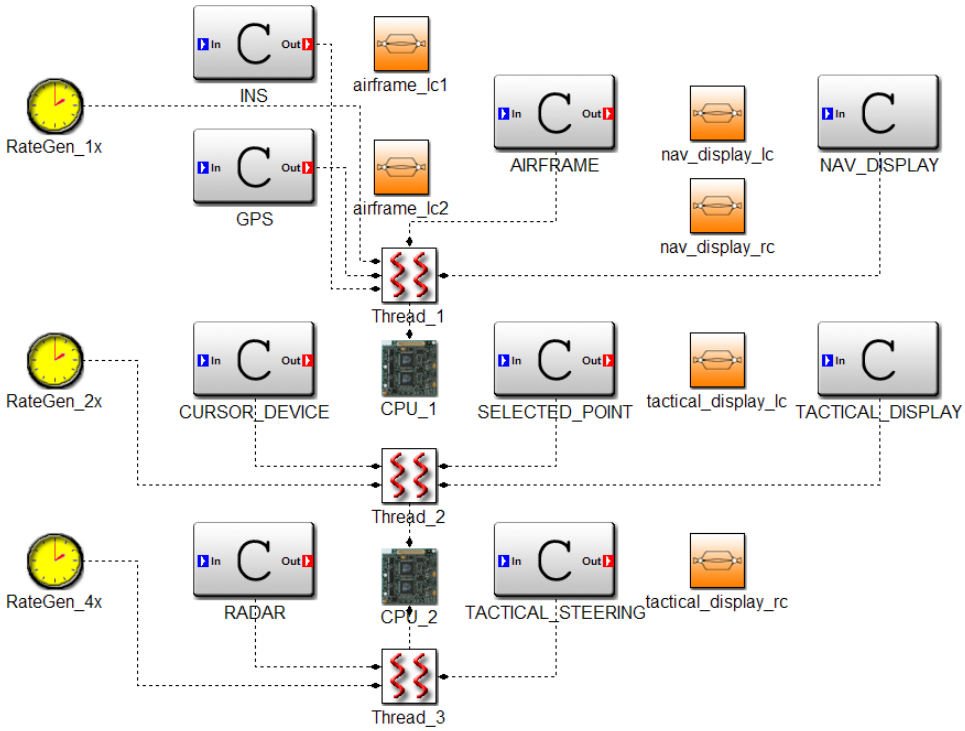

Figure 7: Small Distributed Preemptive Real-time CORBA Application

Task	WCET	BCET	DL	SP
INS	32	32	170	1
GPS	29	29	170	1
AIRFRAME	80	80	246	2
NAV_DISPLAY	19	19	532	3
CURSOR_DEVICE	18	18	65	1
SELECTED_POINT	24	24	25	2
TACTICAL_DISPLAY	21	21	80	3
RADAR	12	12	13	1
TACTICAL_STEERING	16	16	17	2

Channel	WCDelay	BCDelay	BufferSize
airframe_lc1	0	0	2
airframe_lc2	0	0	2
tactical_display_lc	0	0	2
nav_display_lc	0	0	3
tactical_display_rc	2	2	2
nav_display_rc	2	2	6

Table 5: Timing Information for the Small Distributed Preemptive Example

distributed non-preemptive example shown in Figure 5, but the three threads are mapped onto two processors, therefore the CPU_2 processor manages two threads following fixed-priority preemptive scheduling. The computation in Thread_3 is driven by a faster clock than in Thread_2, therefore we assign higher priority to Thread_3. This step conforms to rate-monotonic designs.

5.5 Medium Distributed Non-preemptive Real-time CORBA Application

The medium distributed non-preemptive example is deployed onto a distributed 5-processor execution platform.

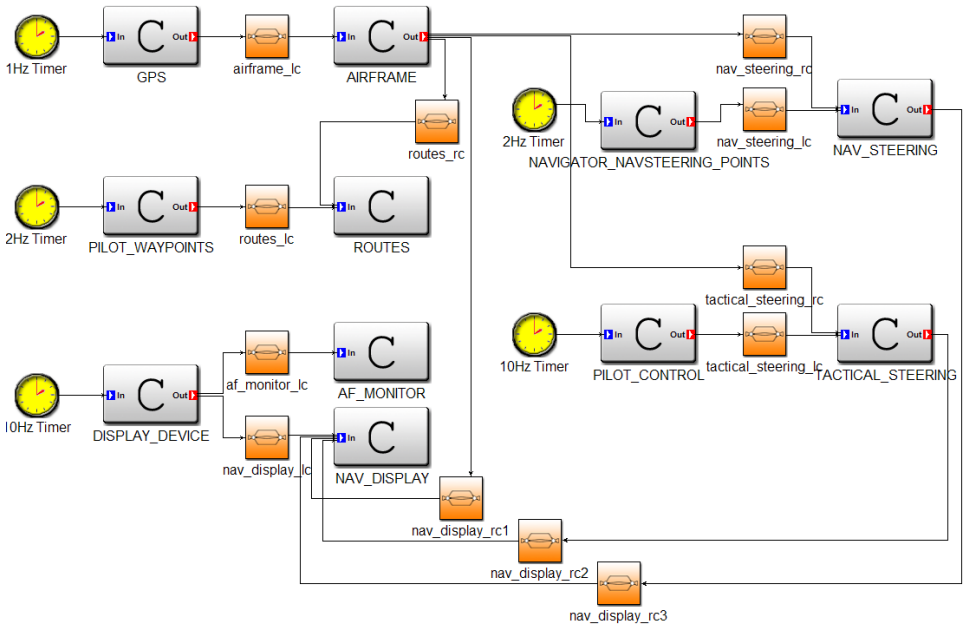

Figure 8: Medium Distributed Non-preemptive Real-time CORBA Application

Task	WCET	BCET	DL	SP
GPS	21	21	22	1
AIRFRAME	53	53	54	2
PILOT_WAYPOINTS	37	37	38	3
ROUTES	18	18	19	4
NAVIGATOR_NAVSTEERING_POINTS	32	32	65	5
NAV_STEERING	49	49	50	6
DISPLAY_DEVICE	26	26	41	7
AF_MONITOR	33	33	34	8
NAV_DISPLAY	14	14	74	9
PILOT_CONTROL	23	23	66	10
TACTICAL_STEERING	58	58	59	11

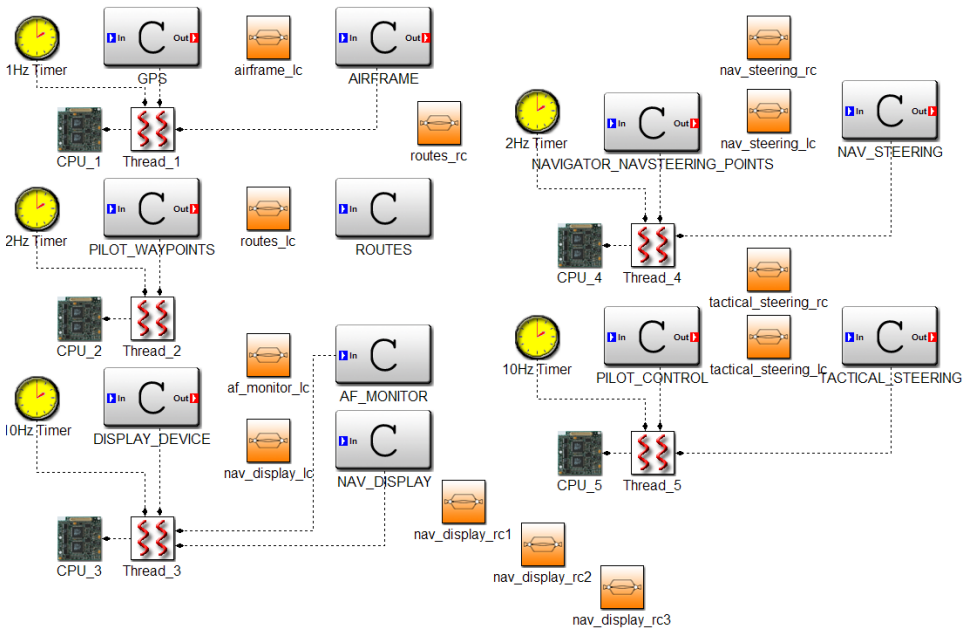

Figure 9: Medium Distributed Non-preemptive Real-time CORBA Application

Channel	WCDelay	BCDelay	BufferSize
nav_steering_lc	0	0	2
routes_lc	0	0	2
tactical_steering_lc	0	0	2
nav_display_lc	0	0	2
af_monitor_lc	0	0	2
nav_steering_rc	2	1	2
routes_rc	2	1	2
tactical_steering_rc	3	1	2
nav_display_rc1	3	1	2
nav_display_rc2	3	1	2
nav_display_rc3	2	1	2

Table 6: Timing Information for the Medium Distributed Non-preemptive Example

5.6 Medium Distributed Preemptive Real-time CORBA Application

The medium distributed preemptive example has the same dependencies as the medium distributed non-preemptive example shown in Figure 8, but is deployed onto a 3-processor preemptive execution platform.

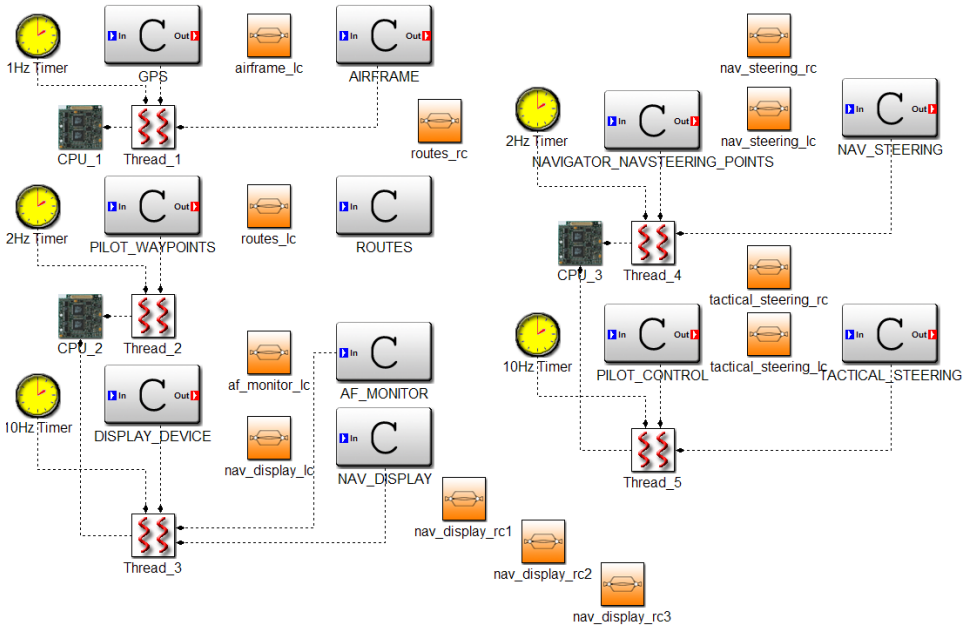

Figure 10: Medium Distributed Preemptive Real-time CORBA Application

Task	WCET	BCET	DL	SP
GPS	21	21	22	1
AIRFRAME	53	53	54	2
PILOT_WAYPOINTS	37	37	38	3
ROUTES	18	18	19	4
NAVIGATOR_NAVSTEERING_POINTS	32	32	65	5
NAV_STEERING	49	49	50	6
DISPLAY_DEVICE	26	26	41	7
AF_MONITOR	33	33	34	8
NAV_DISPLAY	14	14	74	9
PILOT_CONTROL	23	23	66	10
TACTICAL_STEERING	58	58	59	11

Channel	WCDelay	BCDelay	BufferSize
nav_steering_lc	0	0	2
routes_lc	0	0	2
tactical_steering_lc	0	0	2
nav_display_lc	0	0	2
af_monitor_lc	0	0	2
nav_steering_rc	2	1	2
routes_rc	2	1	2
tactical_steering_rc	3	1	2
nav_display_rc1	3	1	2
nav_display_rc2	3	1	2
nav_display_rc3	2	1	2

Table 7: Timing Information for the Medium Distributed Preemptive Example

5.7 Large Distributed Non-preemptive Real-time CORBA Application

We describe a large-scale example to provide a model that can be used to evaluate the scalability of various verification methods. The example is rather large therefore we use two pages to show it. The timers that have the same names on both pages are actually the same timer, separated for illustration purposes only. Tasks that are driven by the same timer are mapped onto the same thread and each thread is assigned to a separate processor.

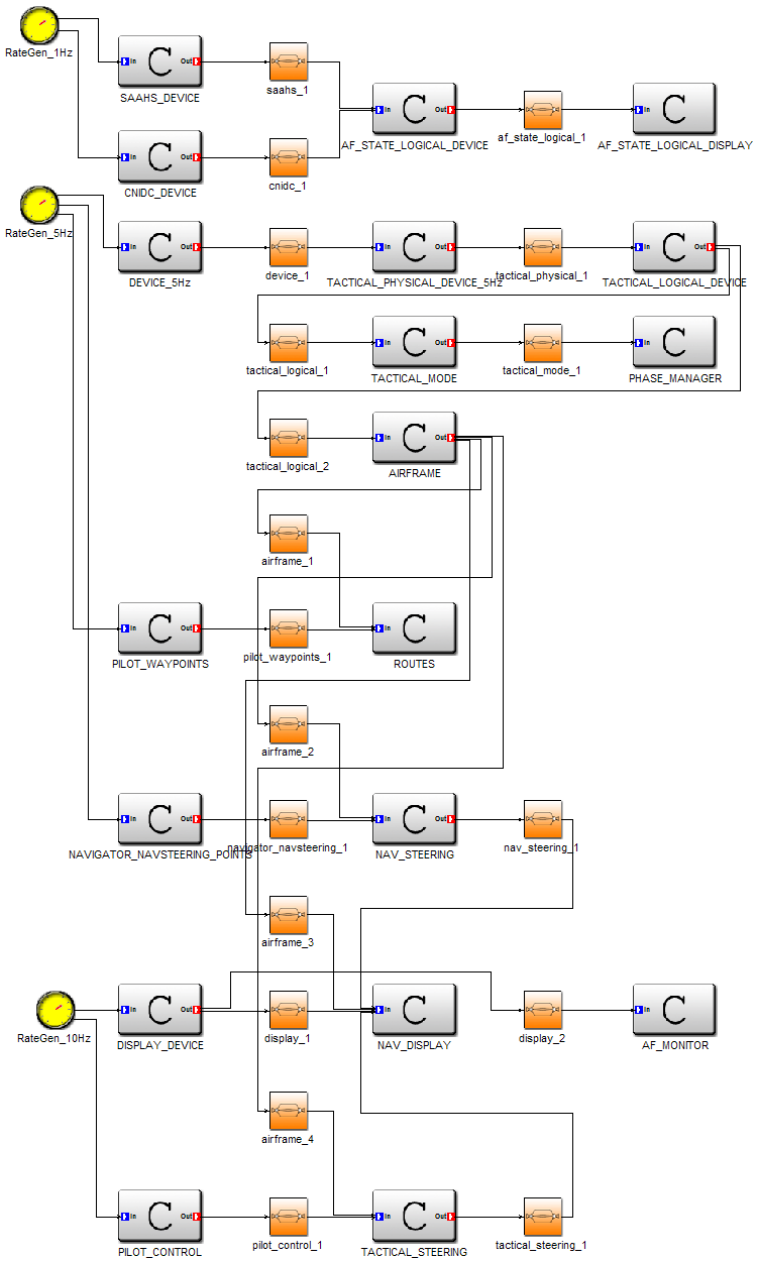

Figure 11: Large Distributed Non-preemptive Real-time CORBA Application (Part I)

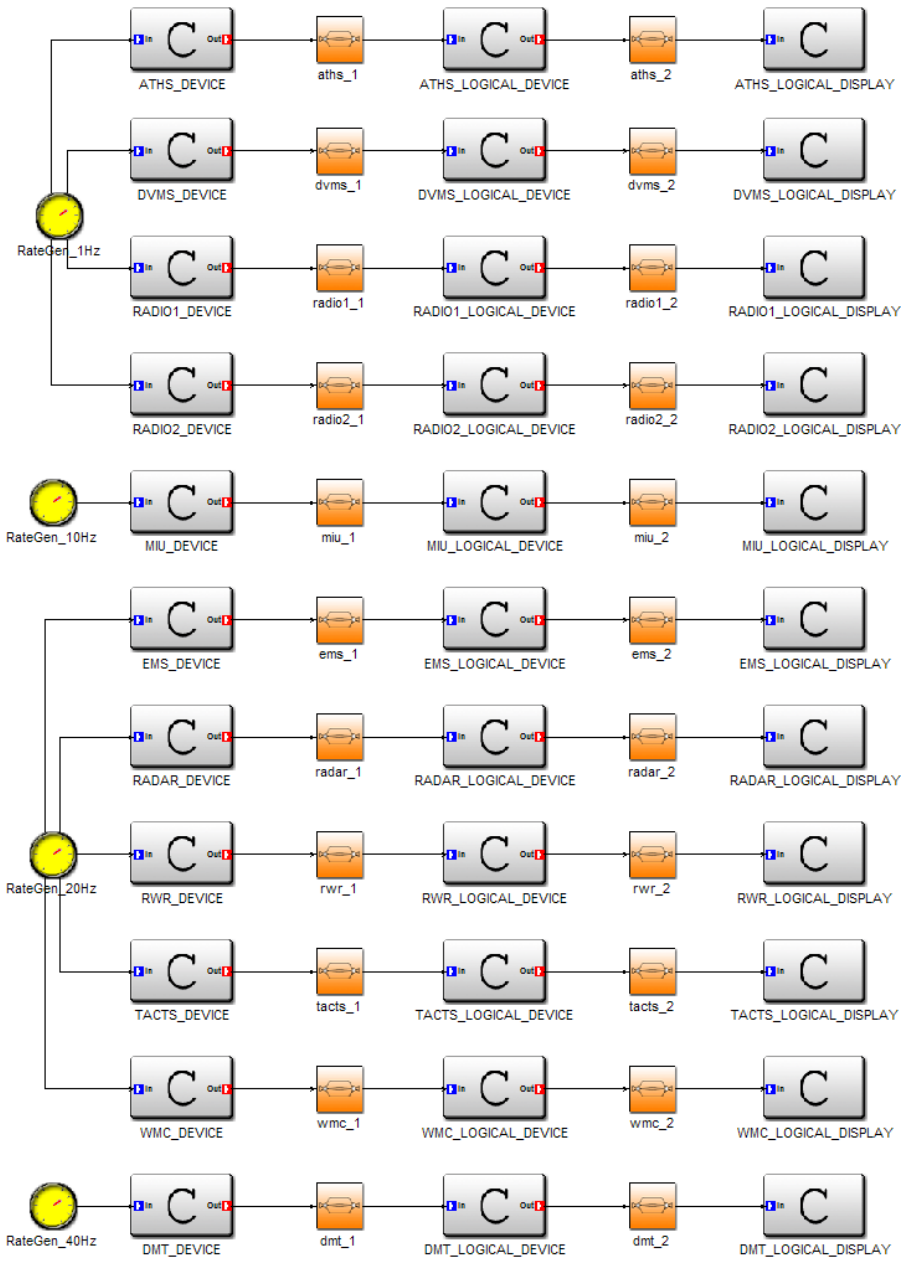

Figure 12: Large Distributed Non-preemptive Real-time CORBA Application (Part II)

6 Timed Automata-based Real-time Verification

This section reviews timed automata as the underlying semantic domain for the proposed analysis that captures both real-time and power-related QoS properties. We chose timed automata since it (1) has well-defined formal semantics [9], and is supported by several automated model checking tools [10, 11, 12], and (2) is expressive enough to capture the dynamics of a wide class of real-time embedded systems, including the formalism presented in Section 4.

A timed automaton is a finite state machine equipped with real-valued clock variables, called *clocks* for short. We use $\mathcal{B}(C)$ to denote the set of atomic clock constraints (or guards) over the set of clocks C where c_i $\mathtt{op}$ n_a or $c_i - c_j$ $\mathtt{op}$ n_b where $C \in \mathcal{R}^+, c_i, c_j \in V, \mathtt{op} \in \{<, \leq, =, \geq, >\}, n_a, n_b \in \mathbb{N}$. Transitions in a timed automaton can have guards and reset operations on clock and data variables. Transitions are enabled if the corresponding guard evaluates to `true`. An enabled transition can execute instantaneously while resetting certain variables to new values according to the underlying reset assignments. States in timed automata can be associated with an invariant that determines the validity of clock assignment in the state. A system can be in a given state only if the underlying invariant is `true`.

Definition 1 *A timed automaton is a 4-tuple $\mathcal{TA} = (L, l_0, E, Inv)$ consisting of the following components:*

- *a finite non-empty set L of vertices called* locations*;*
- *the initial location $l_0 \in L$;*
- *a finite set of edges $E \subseteq L \times \mathcal{B}(C) \times 2^C \times L$ called transitions, where $e = (l, \gamma, \alpha, l')$ represents an edge from location l to location l' with guard γ and α which is an assignment of clocks to the value* zero *(possible clock reset);*
- *a labeling function $Inv : L \rightarrow \mathcal{B}(C)$ that assigns a clock constraint to each location. This constraint is the invariant of the location.* □

Definition 2 *A state of the timed automaton is defined as a pair (l_i, c_i) where $l_i \in L$ and c_i is a valuation of the clock $c_i \in C$. We denote the set of states as S.* □

A more detailed description of the timed automata model can be found in [9]. The Uppaal model checking tool uses an extended version of the above timed automata formulation. The Uppaal extensions to the timed automata model of computation are described in [12].

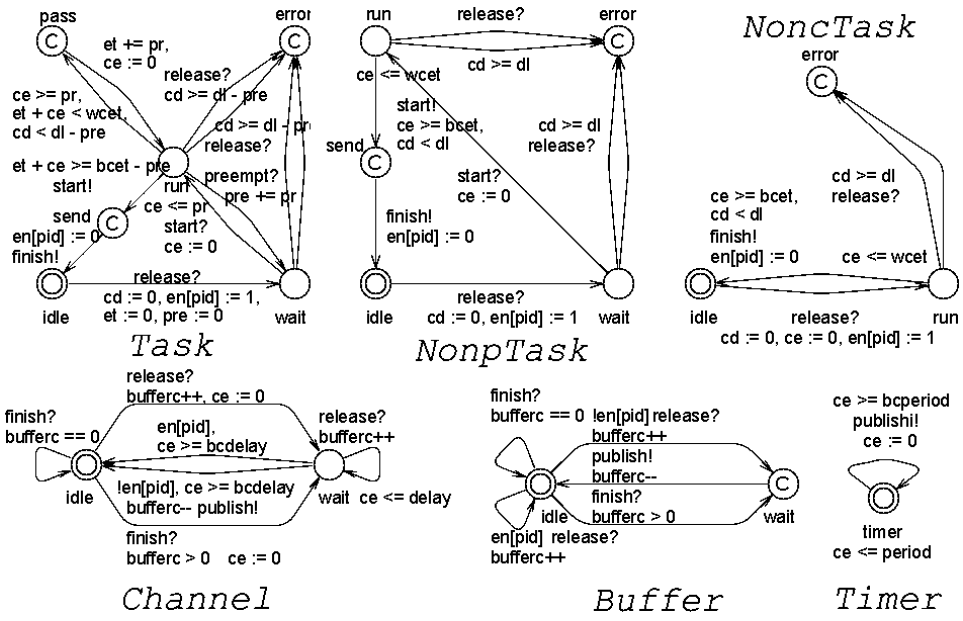

Figure 13: Generic Model of the DRE Semantic Domain

6.1 Timed Automata-based Verification in DREAM

DREAM supports real-time verification based on the timed automata [9] model of computation using the UPPAAL [12] and Verimag IF [10] tools. The timed automata models are automatically extracted from the internal DREAM representation allowing the formal verification of system models.

In this section we show several timed automata models generated from DREAM. The components are captured as task timed automata. Figure 13 shows a parameterizable library of timed automata models that can be composed to model DRE systems. This paper focuses on practical applications of DREAM to several case studies. The theory behind the presented methods is described in [1,13]. Deciding preemptive schedulability is undecidable using timed automata in general [14], but we use a novel conservative approximation method with adjustable precision (pr) to approximate stopwatch automata. Consider the preemptive examples as EXPERIMENTAL.

The fixed-priority scheduler is modeled as an automaton which is composed with the task timed automata. Unfortunately the Verimag IF models do not have a graphical representation. The IF models generated from DREAM are described in the Appendix.

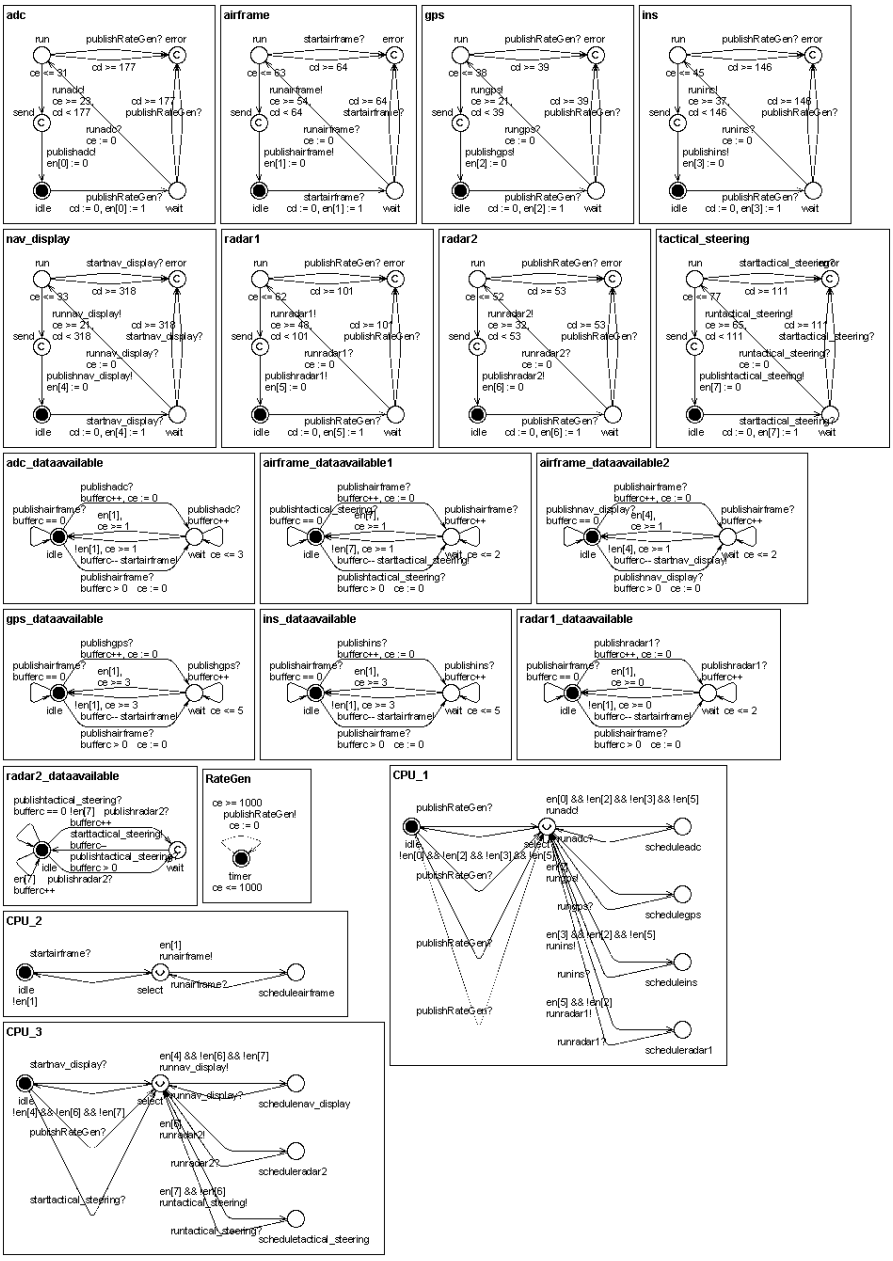

Figure 14: Uppaal Timed Automata Models for the Multiple CPU Non-preemptive Real-time CORBA Application

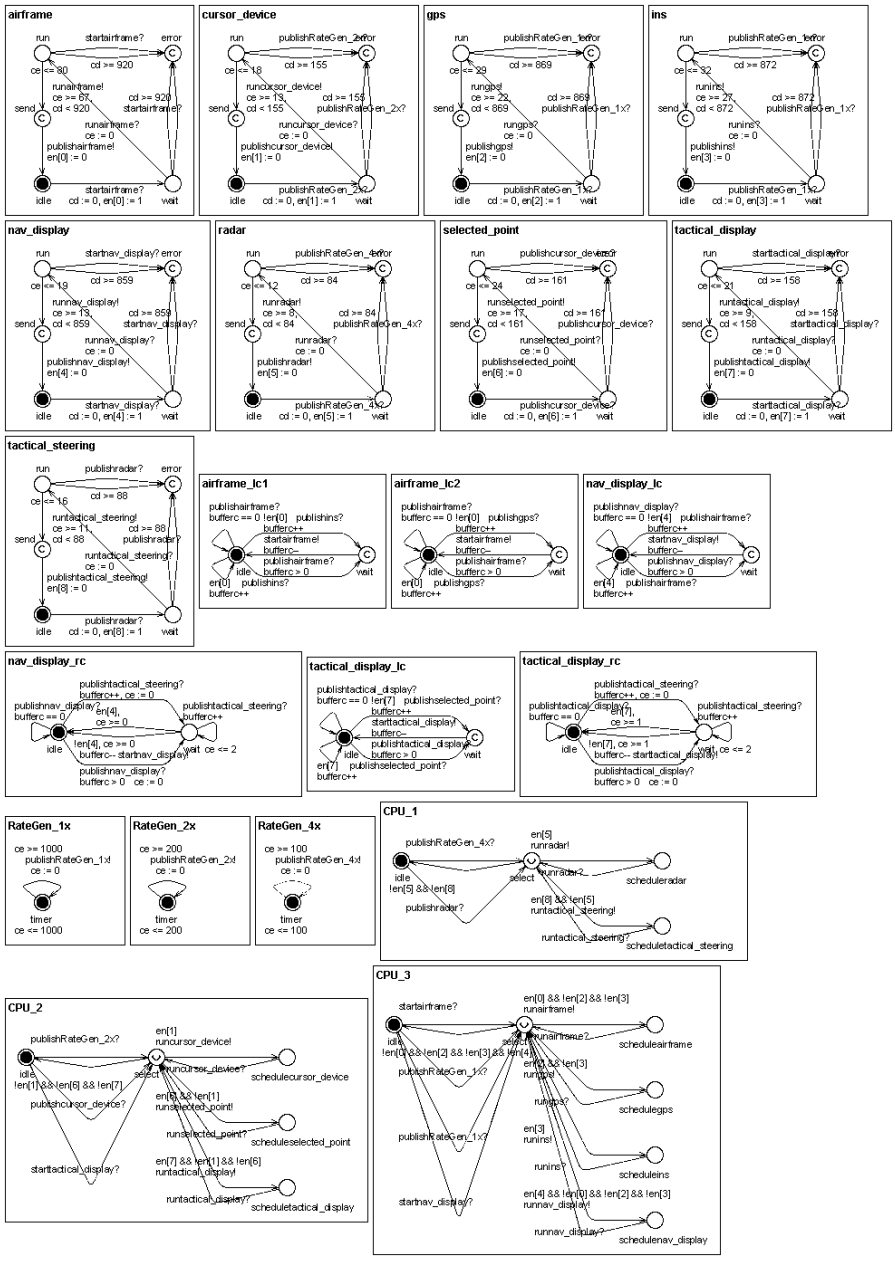

Figure 15: UPPAAL Timed Automata Models for the Small Distributed Non-preemptive Real-time CORBA Application

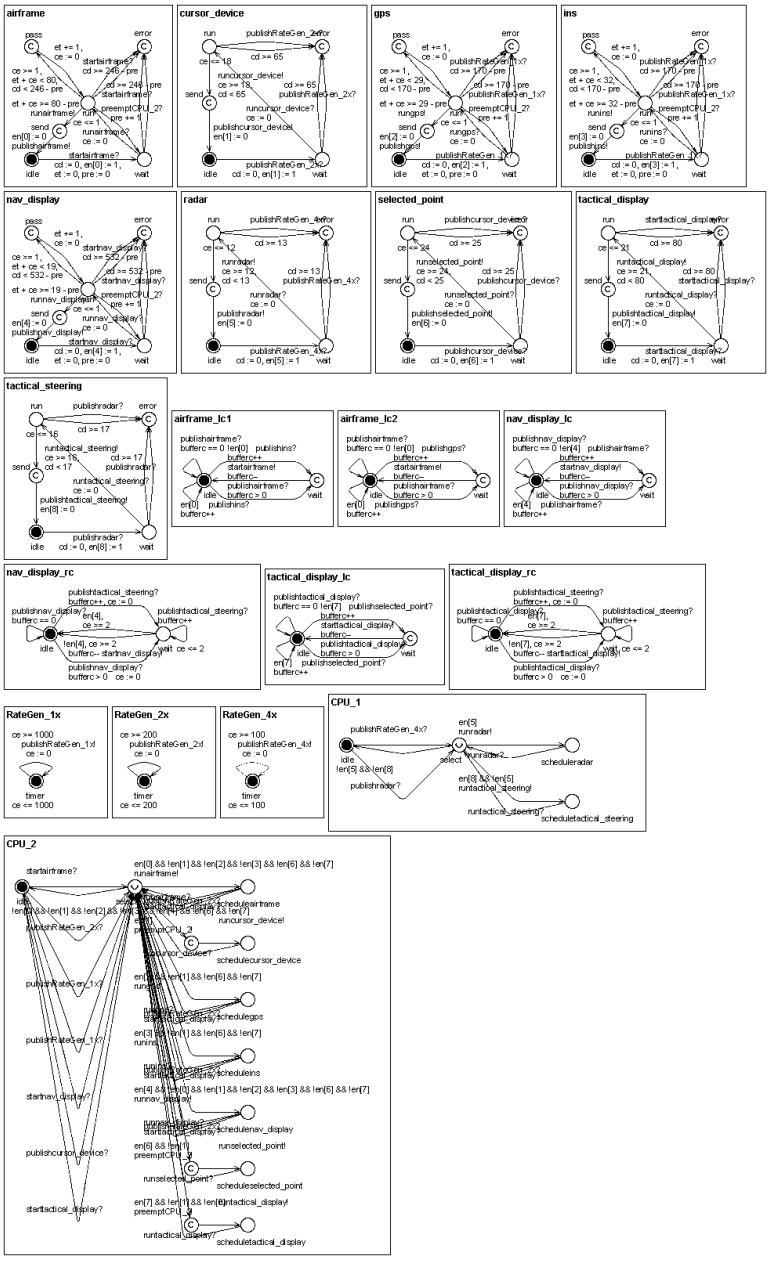

Figure 16: Uppaal Timed Automata Models for the Small Distributed Preemptive Real-time CORBA Application

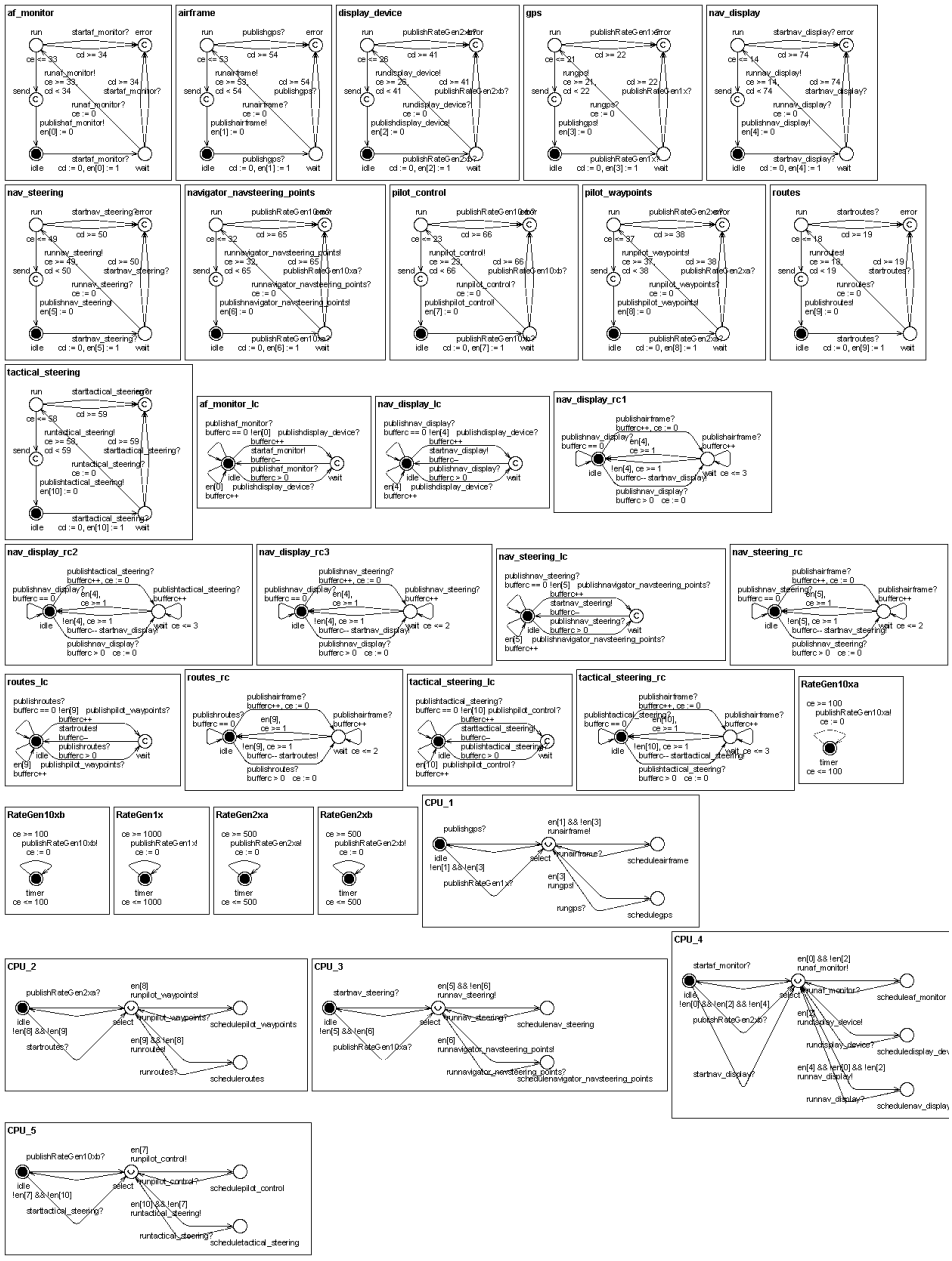

Figure 17: Uppaal Timed Automata Models for the Medium Distributed Non-preemptive Real-time CORBA Application

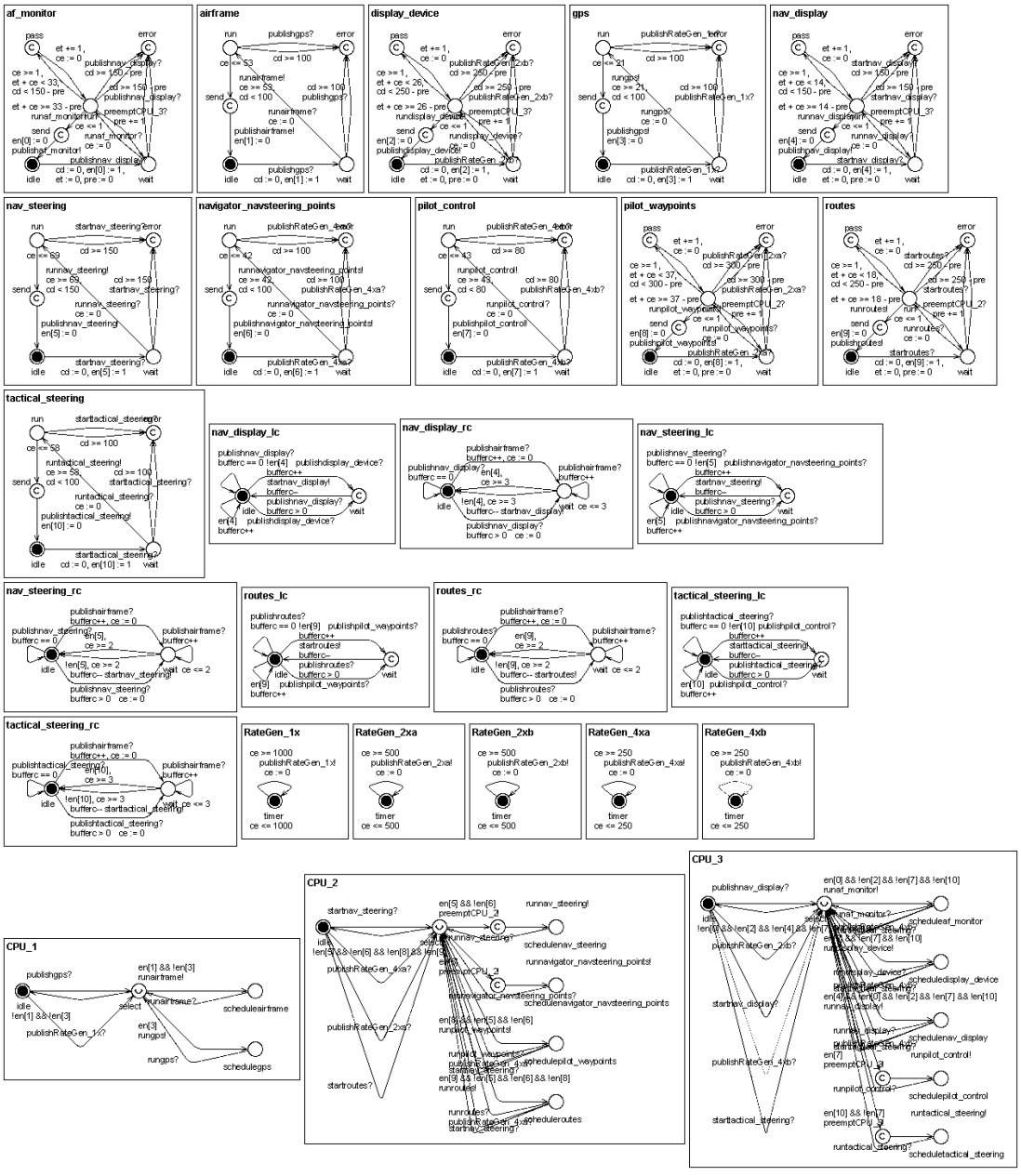

Figure 18: UPPAAL Timed Automata Models for the Medium Distributed Preemptive Real-time CORBA Application

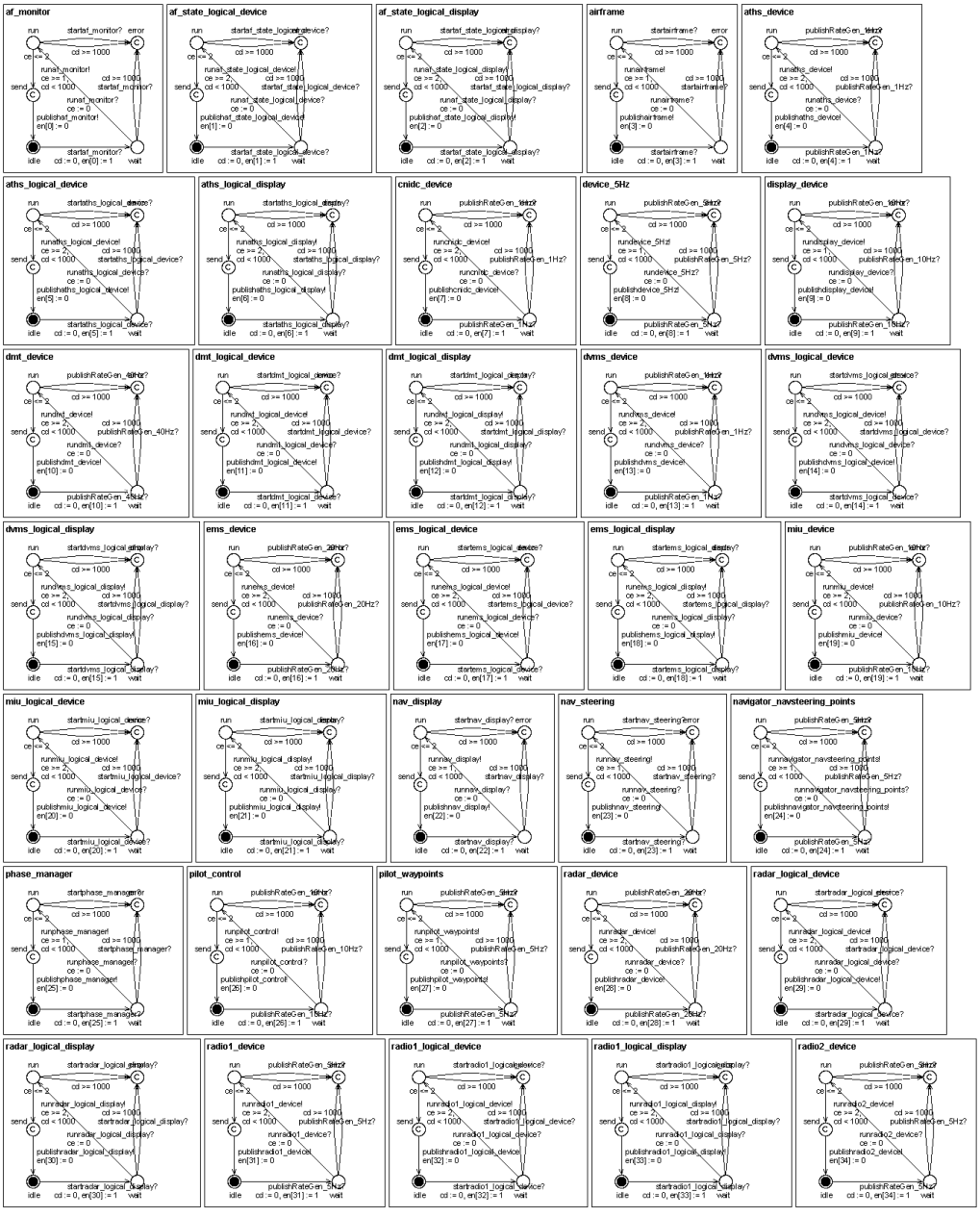

Figure 19: UPPAAL Timed Automata Models for the Large Distributed Non-preemptive Real-time CORBA Application (Part I)

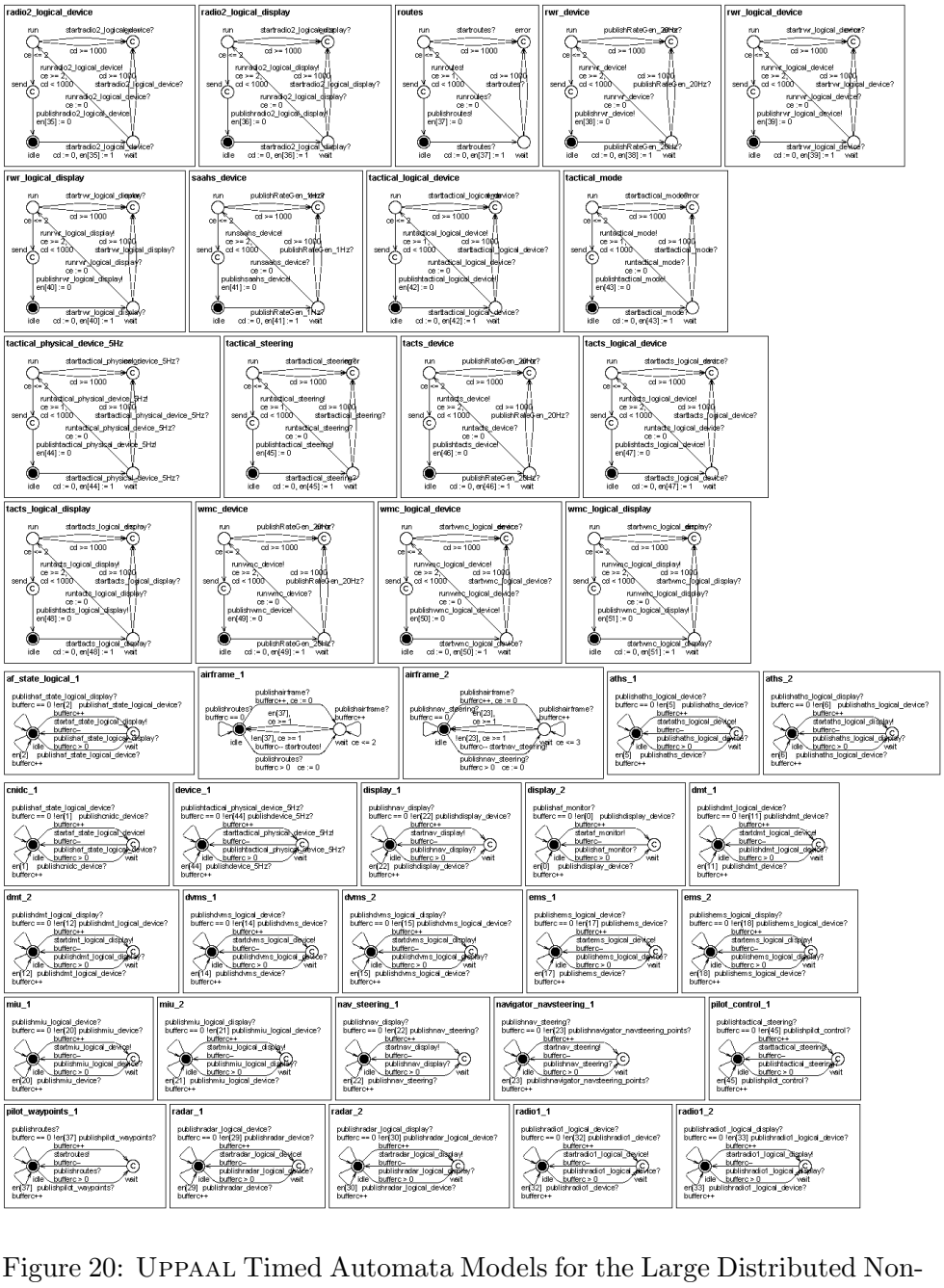

Figure 20: UPPAAL Timed Automata Models for the Large Distributed Non-preemptive Real-time CORBA Application (Part II)

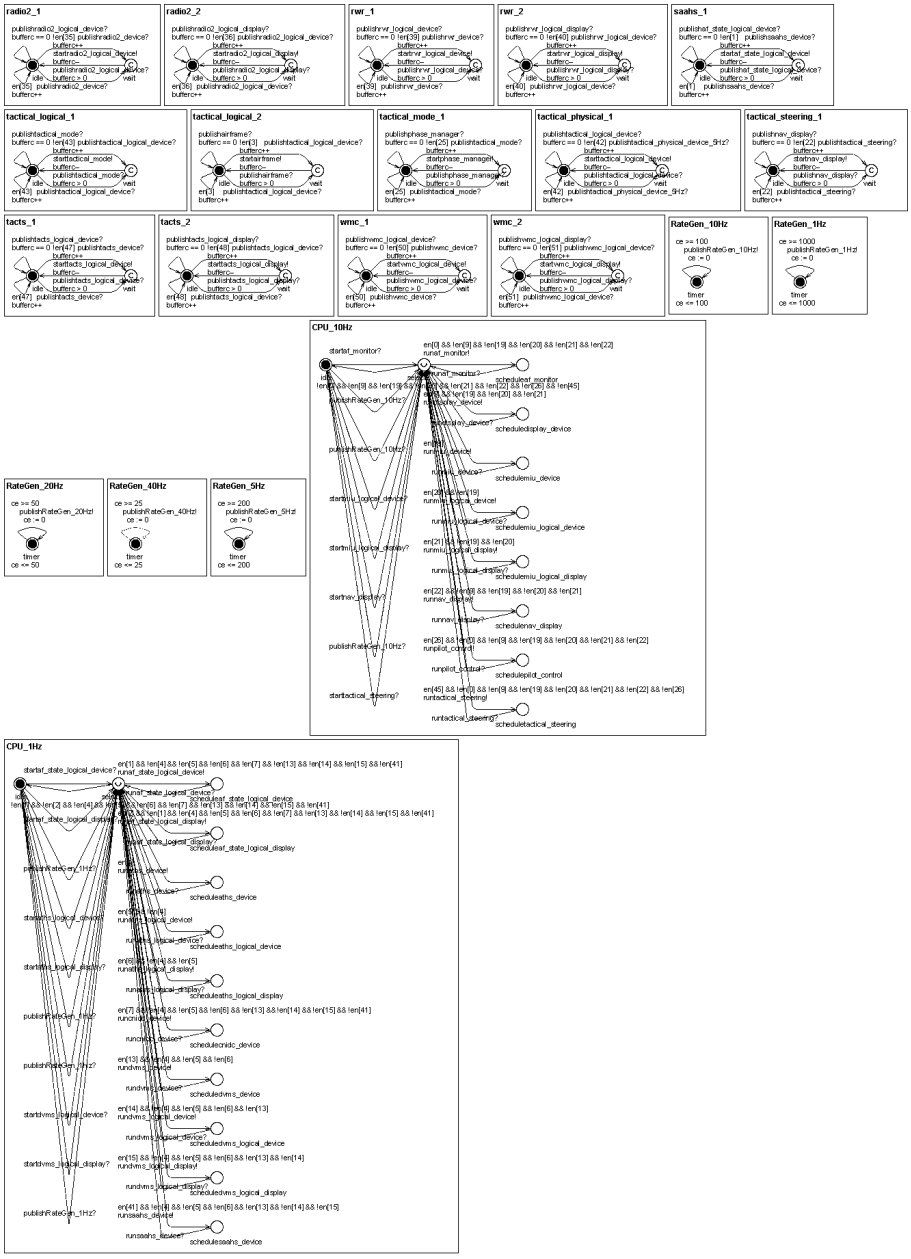

Figure 21: UPPAAL Timed Automata Models for the Large Distributed Non-preemptive Real-time CORBA Application (Part III)

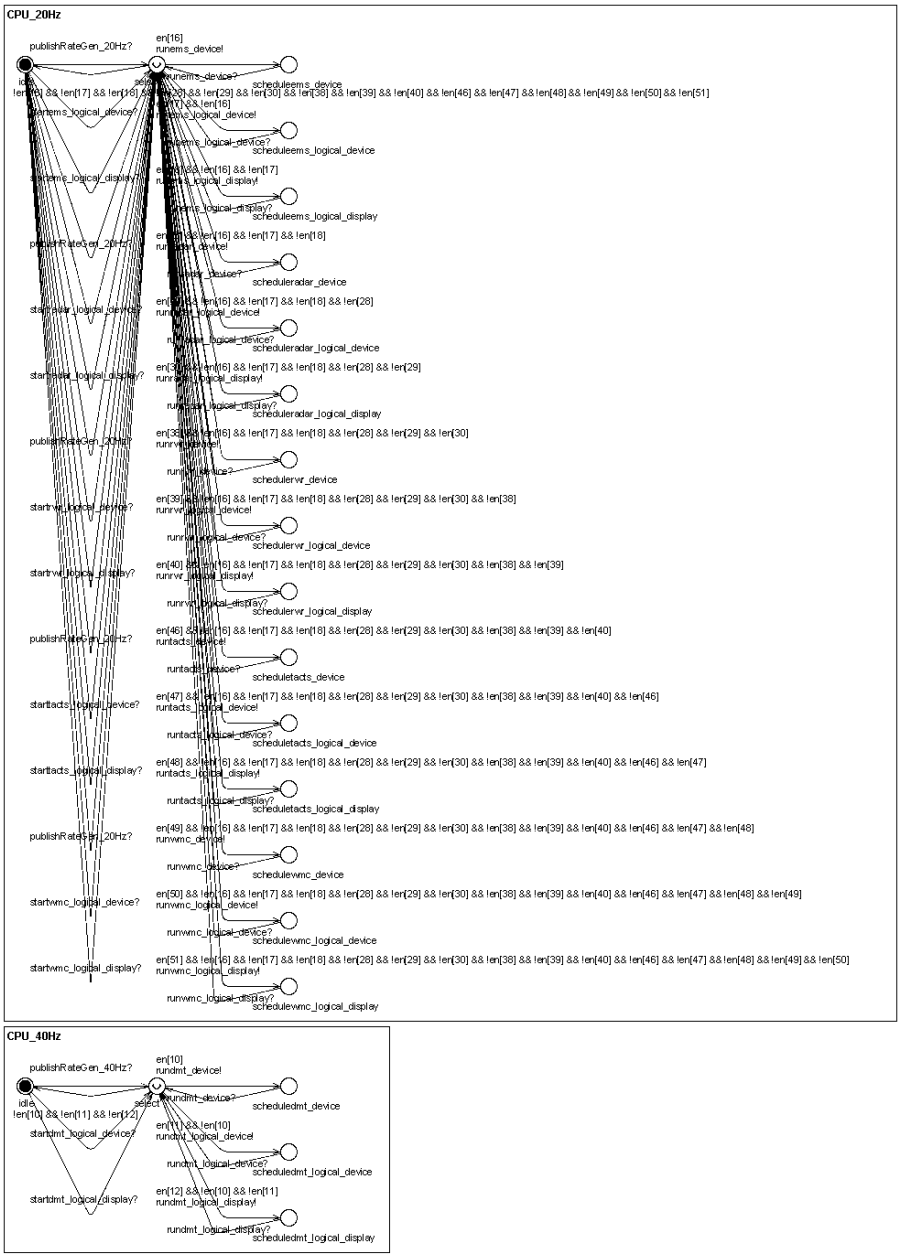

Figure 22: UPPAAL Timed Automata Models for the Large Distributed Non-preemptive Real-time CORBA Application (Part IV)

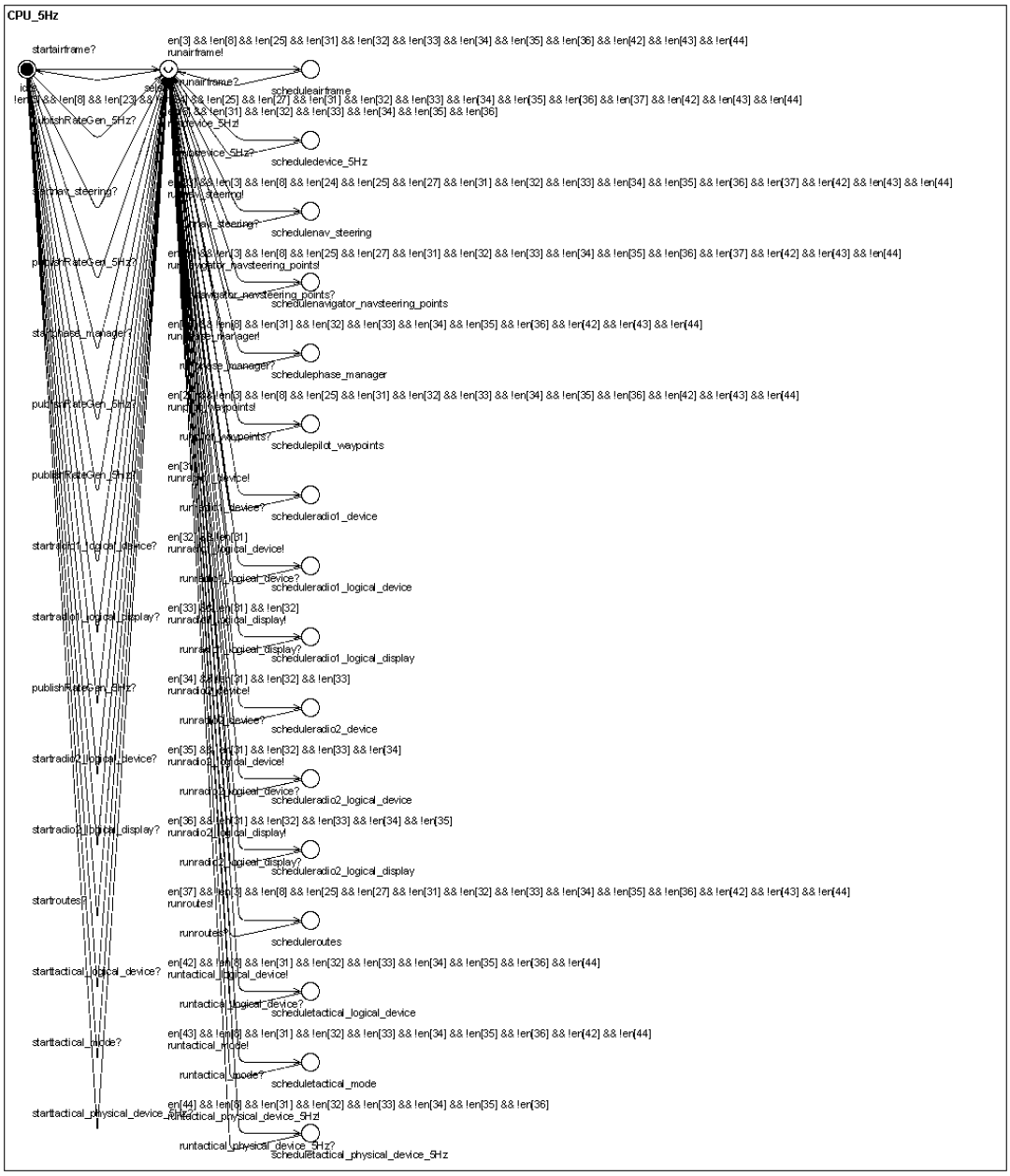

Figure 23: Uppaal Timed Automata Models for the Large Distributed Non-preemptive Real-time CORBA Application (Part V)

7 Simulation-based Verification

DREAM implements a simulation-based model checker for systems using non-preemptive scheduling. Clocks for tasks are represented as real-valued variables that are updated by the discrete event scheduler. The scheduling algorithm is based on a discrete event simulator. The scheduler updates the locations and clocks for the tasks for each event and checks whether a task's execution time is longer than its deadline. Due to some recent results we will include a description of this method and case studies after these results get published.

8 Review of Related Tools

The SAE AADL is an international standard avionics architecture description language. AADL is a successor of the *Honeywell MetaH* toolset [15], a commercially available domain-specific architecture description language (ADL) for developing reliable, real-time multiprocessor avionics system architectures. AADL is a mature platform with a lot of support. DREAM could be extended to operate directly on AADL models, however it would require significant efforts therefore this integration is unlikely to happen anytime soon.

Ptolemy II [16] is a complex modeling framework that composes heterogeneous models of computation to simulate and evaluate embedded systems. Although the MoCs and their composition is formally defined the focus in Ptolemy II is simulation, not verification. In contrast, ALDERIS and the DREAM tool provide a way for formal verification of dense timed systems using several model checkers.

The SYSWEAVER [17] toolset is a component-based framework that supports the reusability of components across systems with different requirements. It supports code generation, as well as automated analysis based on Matlab/Simulink and real-time rate-monotonic analysis tools, such as the TIMEWIZ model-checker. In contrast, ALDERIS focuses on dense time formal verification using the asynchronous event-driven paradigm.

The *Component Synthesis using Model Integrated Computing* (COSMIC) toolkit [18] is an integrated collection of DSMLs that support the development, configuration, deployment, and evaluation of DRE systems based on CIAO, which is an implementation of the CORBA Component Model that is integrated with Real-time CORBA. The major focus of COSMIC is software development, not analysis, and does not support formal verification.

Both CoSMIC and Alderis use the GME tool therefore their integration would be relatively simple.

The *Embedded Systems Modeling Language (ESML)* [19] has been developed to provide a DSML for the Boeing Bold Stroke component-based real-time middleware. Unlike Alderis, ESML is a platform-specific DSML. ESML does not have formal semantics, it is rather a graphical front-end for Bold Stroke software development. Moreover, ESML does not capture power levels or execution intervals. In contrast, Alderis is an analysis language with formal semantics that can be used for the verification of power aware DRE systems. The CoSMIC tools are a more complete and platform-independent solution to provide a graphical language for component-based middleware.

The Cadena [20] framework is an integrated environment for building and analyzing CORBA Component Model (CCM) based systems. Its main functionality includes CCM code generation in Java, dependency analysis, and model-checking. The emphasis of verification in Cadena is on software logical properties. The generated transition system does not support real-time verification in dense time, nor energy savings. In contrast, Alderis represents time and power levels explicitly and allows dense time verification using several model checkers.

The Virginia Embedded Systems Toolkit (VEST) [21] is a framework designed for the reliable and configurable composition and analysis of component-based embedded systems from COTS libraries. VEST applies key checks and analysis but - unlike Alderis - does not support formal proof of correctness.

The FORGE project [22] is a framework for optimization of distributed embedded systems software. Dream is part of the FORGE project and is tailored to analyze power-aware resource-constrained mobile DRE systems.

9 Concluding Remarks

This paper highlights the open-source Dream tool that focuses on the practical application of formal methods to distributed real-time embedded systems. For more information on Dream and downloads please visit `http://dre.sourceforge.net`.

References

[1] G. Madl, S. Abdelwahed, and D. C. Schmidt, "Verifying Distributed Real-time Properties of Embedded Systems via Graph Transforma-

tions and Model Checking (invited paper, accepted)," *The International Journal of Time-Critical Computing*, 2005.

[2] J. Sztipanovits and G. Karsai, "Model-Integrated Computing," *IEEE Computer*, pp. 110–112, Apr. 1997.

[3] Sun Microsystems, "Java 2 Platform, Enterprise Edition (J2EE) v1.4," 2003. [Online]. Available: http://java.sun.com/j2ee

[4] Microsoft, "Microsoft .NET," 2000. [Online]. Available: http://microsoft.com/net

[5] Object Management Group, "CORBA Component Model," 2002. [Online]. Available: http://www.omg.org

[6] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, and J. Sprinkle, "Composing Domain-Specific Design Environments," *Computer*, pp. 44–51, Nov 2001.

[7] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. New York: Wiley & Sons, 2000.

[8] A. B. Arulanthu, C. O'Ryan, D. C. Schmidt, M. Kircher, and J. Parsons, "The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging," in *Proceedings of the Middleware 2000 Conference*. ACM/IFIP, Apr. 2000.

[9] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[10] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, "The IF Toolset," *Formal Methods for the Design of Real-Time Systems, LNCS 3185*, pp. 237–267, Sep 2004.

[11] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The tool KRONOS," in *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*. Springer-Verlag New York, Inc., 1996, pp. 208–219.

[12] P. Pettersson and K. G. Larsen., "Uppaal2k," *Bulletin of the European Association for Theoretical Computer Science*, vol. 70, pp. 40–44, feb 2000.

[13] G. Madl and S. Abdelwahed, "Model-based Analysis of Distributed Real-time Embedded System Composition," in *Proceedings of EMSOFT*, 2005.

[14] P. Krčál and W. Yi, "Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata," in *Proc. of TACAS'04, Barcelona, Spain.*, ser. Lecture Notes in Computer Science, K. Jensen and A. Podelski, Eds., vol. 2988. Springer–Verlag, 2004, pp. 236–250.

[15] S. Vestal, "Formal Verification of the MetaH Executive Using Linear Hybrid Automata," in *RTAS '00: Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000).* Washington, DC, USA: IEEE Computer Society, 2000, p. 134.

[16] E. A. Lee, C. Hylands, J. Janneck, J. D. II, J. Liu, X. Liu, S. Neuendorffer, S. S. M. Stewart, K. Vissers, and P. Whitaker, "Overview of the ptolemy project," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M01/11, 2001.

[17] D. de Niz, G. Bhatia, and R. Rajkumar, "Model-Based Development of Embedded Systems: The SysWeaver Approach," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, 2006, pp. 231–242.

[18] A. Gokhale, K. Balasubramanian, J. Balasubramanian, A. S. Krishna, G. T. Edwards, G. Deng, E. Turkay, J. Parsons, and D. C. Schmidt, "Model Driven Middleware: A New Paradigm for Deploying and Provisioning Distributed Real-time and Embedded Applications," *The Journal of Science of Computer Programming: Special Issue on Model Driven Architecture*, 2005 (to appear).

[19] G. Karsai, S. Neema, A. Bakay, A. Ledeczi, F. Shi, and A. Gokhale, "A Model-based Front-end to TAO/ACE," in *Proceedings of the 2nd Workshop on TAO*, 2002.

[20] J. Hatcliff, X. Deng, M. B. Dwyer, G. Jung, and V. P. Ranganath, "Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems," in *Proceedings of International Conference on Software Engineering*, 2003.

[21] J.A. Stankovic and R. Zhu and R. Poornalingham and C. Lu and Z. Yu and M. Humphrey and B. Ellis, "VEST: An Aspect-based Composi-

tion Tool for Real-time Systems," in *Proceedings of the IEEE Real-time Applications Symposium*, 2003.

[22] R. Cornea, N. Dutt, R. Gupta, I. Krueger, A. Nicolau, D. Schmidt, and S. Shukla, "FORGE: A Framework for Optimization of Distributed Embedded Systems Software," in *Proceedings of the International Parallel and Distributed Processing Symposium*, April 2003.

10 Appendix

The following code fragments are shown for illustration purposes only. Please refer to the DREAM website `http://dre.sourceforge.net` for copyright information.

10.1 Single CPU Non-preemptive ALDERIS XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../xsd/dream.xsd" name="single_cpu_nonp" version="0.4">
<DependencyAspect>
<Node>
<Task name="gps" bcet="21" deadline="168" subpriority="2" wcet="38"/>
<Task name="ins" bcet="37" deadline="621" subpriority="6" wcet="45"/>
<Task name="adc" bcet="23" deadline="825" subpriority="7" wcet="31"/>
<Task name="radar1" bcet="48" deadline="370" subpriority="4" wcet="62"/>
<Task name="radar2" bcet="32" deadline="53" subpriority="0" wcet="52"/>
<Task name="airframe" bcet="54" deadline="64" subpriority="3" wcet="63"/>
<Task name="nav_display" bcet="21" deadline="313" subpriority="5" wcet="33"/>
<Task name="tactical_steering" bcet="65" deadline="78" subpriority="1" wcet="77"/>
<Channel name="gps_dataavailable" buffersize="2"/>
<Channel name="ins_dataavailable" buffersize="2"/>
<Channel name="adc_dataavailable" buffersize="2"/>
<Channel name="radar1_dataavailable" buffersize="2"/>
<Channel name="radar2_dataavailable" buffersize="2"/>
<Channel name="airframe_dataavailable1" buffersize="2"/>
<Channel name="airframe_dataavailable2" buffersize="2"/>
<Timer name="RateGen" period="1000"/>
</Node>
<Hierarchy>
<Component name="GPS">
<TaskContainment task="gps"/>
</Component>
<Component name="INS">
<TaskContainment task="ins"/>
</Component>
<Component name="ADC">
<TaskContainment task="adc"/>
</Component>
<Component name="RADAR1">
<TaskContainment task="radar1"/>
```

```
</Component>
<Component name="RADAR2">
<TaskContainment task="radar2"/>
</Component>
<Component name="AIRFRAME">
<TaskContainment task="airframe"/>
</Component>
<Component name="NAV_DISPLAY">
<TaskContainment task="nav_display"/>
</Component>
<Component name="TACTICAL_STEERING">
<TaskContainment task="tactical_steering"/>
</Component>
</Hierarchy>
<Dependency>
<TimerToTask timer="RateGen" task="ins"/>
<TimerToTask timer="RateGen" task="gps"/>
<TimerToTask timer="RateGen" task="adc"/>
<TimerToTask timer="RateGen" task="radar1"/>
<TimerToTask timer="RateGen" task="radar2"/>
<TaskToChannel task="gps" channel="gps_dataavailable"/>
<TaskToChannel task="ins" channel="ins_dataavailable"/>
<TaskToChannel task="adc" channel="adc_dataavailable"/>
<TaskToChannel task="radar1" channel="radar1_dataavailable"/>
<TaskToChannel task="radar2" channel="radar2_dataavailable"/>
<TaskToChannel task="airframe" channel="airframe_dataavailable1"/>
<TaskToChannel task="airframe" channel="airframe_dataavailable2"/>
<ChannelToTask channel="gps_dataavailable" task="airframe"/>
<ChannelToTask channel="ins_dataavailable" task="airframe"/>
<ChannelToTask channel="adc_dataavailable" task="airframe"/>
<ChannelToTask channel="radar1_dataavailable" task="airframe"/>
<ChannelToTask channel="radar2_dataavailable" task="tactical_steering"/>
<ChannelToTask channel="airframe_dataavailable1" task="tactical_steering"/>
<ChannelToTask channel="airframe_dataavailable2" task="nav_display"/>
</Dependency>
</DependencyAspect>
<PlatformAspect>
<Thread name="Thread" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="gps"/>
<TaskMapping task="ins"/>
<TaskMapping task="adc"/>
<TaskMapping task="radar1"/>
<TaskMapping task="radar2"/>
<TaskMapping task="airframe"/>
<TaskMapping task="tactical_steering"/>
<TaskMapping task="nav_display"/>
<TimerMapping timer="RateGen"/>
</Thread>
<Thread name="Channels" queueingpolicy="FixedPriority">
<ChannelMapping channel="gps_dataavailable"/>
<ChannelMapping channel="ins_dataavailable"/>
<ChannelMapping channel="adc_dataavailable"/>
<ChannelMapping channel="radar1_dataavailable"/>
<ChannelMapping channel="radar2_dataavailable"/>
<ChannelMapping channel="airframe_dataavailable1"/>
<ChannelMapping channel="airframe_dataavailable2"/>
```

```
</Thread>
<CPU name="CPU_1" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread"/>
</CPU>
<CPU name="ChannelManager" schedulingpolicy="NonConcurrent">
<ThreadMapping thread="Channels"/>
</CPU>
</PlatformAspect>
</DRESystem>
```

10.2 Multiple CPU Non-preemptive ALDERIS XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../xsd/dream.xsd" name="multiple_cpu_nonp" version="0.4">
<DependencyAspect>
<Node>
<Task name="gps" bcet="21" deadline="39" subpriority="2" wcet="38"/>
<Task name="ins" bcet="37" deadline="146" subpriority="6" wcet="45"/>
<Task name="adc" bcet="23" deadline="177" subpriority="7" wcet="31"/>
<Task name="radar1" bcet="48" deadline="101" subpriority="4" wcet="62"/>
<Task name="radar2" bcet="32" deadline="53" subpriority="0" wcet="52"/>
<Task name="airframe" bcet="54" deadline="64" subpriority="3" wcet="63"/>
<Task name="nav_display" bcet="21" deadline="318" subpriority="5" wcet="33"/>
<Task name="tactical_steering" bcet="65" deadline="111" subpriority="1" wcet="77"/>
<Channel name="gps_dataavailable" buffersize="2" bcdelay="3" delay="5"/>
<Channel name="ins_dataavailable" buffersize="2" bcdelay="3" delay="5"/>
<Channel name="adc_dataavailable" buffersize="2" bcdelay="1" delay="3"/>
<Channel name="radar1_dataavailable" buffersize="2" bcdelay="0" delay="2"/>
<Channel name="radar2_dataavailable" buffersize="2"/>
<Channel name="airframe_dataavailable1" bcdelay="1" delay="2" buffersize="3"/>
<Channel name="airframe_dataavailable2" bcdelay="1" delay="2" buffersize="4"/>
<Timer name="RateGen" period="1000"/>
</Node>
<Hierarchy>
<Component name="GPS">
<TaskContainment task="gps"/>
</Component>
<Component name="INS">
<TaskContainment task="ins"/>
</Component>
<Component name="ADC">
<TaskContainment task="adc"/>
</Component>
<Component name="RADAR1">
<TaskContainment task="radar1"/>
</Component>
<Component name="RADAR2">
<TaskContainment task="radar2"/>
</Component>
<Component name="AIRFRAME">
<TaskContainment task="airframe"/>
</Component>
<Component name="NAV_DISPLAY">
<TaskContainment task="nav_display"/>
```

```
</Component>
<Component name="TACTICAL_STEERING">
<TaskContainment task="tactical_steering"/>
</Component>
</Hierarchy>
<Dependency>
<TimerToTask timer="RateGen" task="ins"/>
<TimerToTask timer="RateGen" task="gps"/>
<TimerToTask timer="RateGen" task="adc"/>
<TimerToTask timer="RateGen" task="radar1"/>
<TimerToTask timer="RateGen" task="radar2"/>
<TaskToChannel task="gps" channel="gps_dataavailable"/>
<TaskToChannel task="ins" channel="ins_dataavailable"/>
<TaskToChannel task="adc" channel="adc_dataavailable"/>
<TaskToChannel task="radar1" channel="radar1_dataavailable"/>
<TaskToChannel task="radar2" channel="radar2_dataavailable"/>
<TaskToChannel task="airframe" channel="airframe_dataavailable1"/>
<TaskToChannel task="airframe" channel="airframe_dataavailable2"/>
<ChannelToTask channel="gps_dataavailable" task="airframe"/>
<ChannelToTask channel="ins_dataavailable" task="airframe"/>
<ChannelToTask channel="adc_dataavailable" task="airframe"/>
<ChannelToTask channel="radar1_dataavailable" task="airframe"/>
<ChannelToTask channel="radar2_dataavailable" task="tactical_steering"/>
<ChannelToTask channel="airframe_dataavailable1" task="tactical_steering"/>
<ChannelToTask channel="airframe_dataavailable2" task="nav_display"/>

    </Dependency>
</DependencyAspect>
<PlatformAspect>
<Thread name="Thread_1" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="gps"/>
<TaskMapping task="ins"/>
<TaskMapping task="adc"/>
<TaskMapping task="radar1"/>
<TimerMapping timer="RateGen"/>
</Thread>
<Thread name="Thread_2" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="airframe"/>
</Thread>
<Thread name="Thread_3" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="nav_display"/>
<TaskMapping task="radar2"/>
<TaskMapping task="tactical_steering"/>
</Thread>
<Thread name="Channels" queueingpolicy="FixedPriority">
<ChannelMapping channel="gps_dataavailable"/>
<ChannelMapping channel="ins_dataavailable"/>
<ChannelMapping channel="adc_dataavailable"/>
<ChannelMapping channel="radar1_dataavailable"/>
<ChannelMapping channel="radar2_dataavailable"/>
<ChannelMapping channel="airframe_dataavailable1"/>
<ChannelMapping channel="airframe_dataavailable2"/>
</Thread>
<CPU name="CPU_1" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_1"/>
</CPU>
```

```
<CPU name="CPU_2" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_2"/>
</CPU>
<CPU name="CPU_3" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_3"/>
</CPU>
<CPU name="ChannelManager" schedulingpolicy="NonConcurrent">
<ThreadMapping thread="Channels"/>
</CPU>
</PlatformAspect>
</DRESystem>
```

10.3 Small Distributed Non-preemptive ALDERIS XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../xsd/dream.xsd" name="small_distributed_nonp"
version="0.3">
<DependencyAspect>
<Node>
<Task name="radar" bcet="8" deadline="84" subpriority="1" wcet="12"/>
<Task name="tactical_steering" bcet="11" deadline="88" subpriority="2" wcet="16"/>
<Task name="cursor_device" bcet="13" deadline="155" subpriority="3" wcet="18"/>
<Task name="selected_point" bcet="17" deadline="161" subpriority="4" wcet="24"/>
<Task name="tactical_display" bcet="9" deadline="158" subpriority="5" wcet="21"/>
<Task name="ins" bcet="27" deadline="872" subpriority="6" wcet="32"/>
<Task name="gps" bcet="22" deadline="869" subpriority="7" wcet="29"/>
<Task name="airframe" bcet="67" deadline="920" subpriority="8" wcet="80"/>
<Task name="nav_display" bcet="13" deadline="859" subpriority="9" wcet="19"/>
<Channel name="airframe_lc1" buffersize="2"/>
<Channel name="airframe_lc2" buffersize="2"/>
<Channel name="tactical_display_lc" buffersize="2"/>
<Channel name="nav_display_lc" buffersize="3"/>
<Channel name="tactical_display_rc" buffersize="2" bcdelay="1" delay="2"/>
<Channel name="nav_display_rc" buffersize="5" bcdelay="0" delay="2"/>
<Timer name="RateGen_1x" period="1000"/>
<Timer name="RateGen_2x" period="200"/>
<Timer name="RateGen_4x" period="100"/>
</Node>
<Hierarchy>
<Component name="RADAR">
<TaskContainment task="radar"/>
</Component>
<Component name="TACTICAL_STEERING">
<TaskContainment task="tactical_steering"/>
</Component>
<Component name="CURSOR_DEVICE">
<TaskContainment task="cursor_device"/>
</Component>
<Component name="SELECTED_POINT">
<TaskContainment task="selected_point"/>
</Component>
<Component name="TACTICAL_DISPLAY">
<TaskContainment task="tactical_display"/>
</Component>
```

```
<Component name="INS">
<TaskContainment task="ins"/>
</Component>
<Component name="GPS">
<TaskContainment task="gps"/>
</Component>
<Component name="AIRFRAME">
<TaskContainment task="airframe"/>
</Component>
<Component name="NAV_DISPLAY">
<TaskContainment task="nav_display"/>
</Component>
</Hierarchy>
<Dependency>
<TimerToTask timer="RateGen_1x" task="ins"/>
<TimerToTask timer="RateGen_1x" task="gps"/>
<TimerToTask timer="RateGen_2x" task="cursor_device"/>
<TimerToTask timer="RateGen_4x" task="radar"/>
<TaskToTask sourcetask="cursor_device" destinationtask="selected_point"/>
<TaskToTask sourcetask="radar" destinationtask="tactical_steering"/>
<TaskToChannel task="ins" channel="airframe_lc1"/>
<TaskToChannel task="gps" channel="airframe_lc2"/>
<TaskToChannel task="airframe" channel="nav_display_lc"/>
<TaskToChannel task="selected_point" channel="tactical_display_lc"/>
<TaskToChannel task="tactical_steering" channel="tactical_display_rc"/>
<TaskToChannel task="tactical_steering" channel="nav_display_rc"/>
<ChannelToTask channel="airframe_lc1" task="airframe"/>
<ChannelToTask channel="airframe_lc2" task="airframe"/>
<ChannelToTask channel="tactical_display_lc" task="tactical_display"/>
<ChannelToTask channel="nav_display_lc" task="nav_display"/>
<ChannelToTask channel="tactical_display_rc" task="tactical_display"/>
<ChannelToTask channel="nav_display_rc" task="nav_display"/>
</Dependency>
</DependencyAspect>
<PlatformAspect>
<Thread name="Thread_1" queueingpolicy="FixedPriority">
<TaskMapping task="radar"/>
<TaskMapping task="tactical_steering"/>
<TimerMapping timer="RateGen_4x"/>
</Thread>
<Thread name="Thread_2" queueingpolicy="FixedPriority">
<TaskMapping task="cursor_device"/>
<TaskMapping task="selected_point"/>
<TaskMapping task="tactical_display"/>
<TimerMapping timer="RateGen_2x"/>
</Thread>
<Thread name="Thread_3" queueingpolicy="FixedPriority">
<TaskMapping task="ins"/>
<TaskMapping task="gps"/>
<TaskMapping task="airframe"/>
<TaskMapping task="nav_display"/>
<TimerMapping timer="RateGen_1x"/>
</Thread>
<Thread name="Channels" queueingpolicy="FixedPriority">
<ChannelMapping channel="airframe_lc1"/>
<ChannelMapping channel="airframe_lc2"/>
```

```
<ChannelMapping channel="tactical_display_lc"/>
<ChannelMapping channel="nav_display_lc"/>
<ChannelMapping channel="tactical_display_rc"/>
<ChannelMapping channel="nav_display_rc"/>
</Thread>
<CPU name="CPU_1" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_1"/>
</CPU>
<CPU name="CPU_2" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_2"/>
</CPU>
<CPU name="CPU_3" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_3"/>
</CPU>
<CPU name="ChannelManager" schedulingpolicy="NonConcurrent">
<ThreadMapping thread="Channels"/>
</CPU>
</PlatformAspect>
</DRESystem>
```

10.4 Small Distributed Preemptive ALDERIS XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../xsd/dream.xsd" name="small_distributed" version="0.3">
<DependencyAspect>
<Node>
<Task name="radar" deadline="13" subpriority="1" wcet="12"/>
<Task name="tactical_steering" deadline="17" subpriority="2" wcet="16"/>
<Task name="cursor_device" deadline="65" subpriority="1" wcet="18"/>
<Task name="selected_point" deadline="25" subpriority="2" wcet="24"/>
<Task name="tactical_display" deadline="80" subpriority="3" wcet="21"/>
<Task name="ins" deadline="170" subpriority="1" wcet="32"/>
<Task name="gps" deadline="170" subpriority="1" wcet="29"/>
<Task name="airframe" deadline="246" subpriority="2" wcet="80"/>
<Task name="nav_display" deadline="532" subpriority="3" wcet="19"/>
<Channel name="airframe_lc1" buffersize="2"/>
<Channel name="airframe_lc2" buffersize="2"/>
<Channel name="tactical_display_lc" buffersize="2"/>
<Channel name="nav_display_lc" buffersize="3"/>
<Channel name="tactical_display_rc" buffersize="2" delay="2"/>
<Channel name="nav_display_rc" buffersize="6" delay="2"/>
<Timer name="RateGen_1x" period="1000"/>
<Timer name="RateGen_2x" period="200"/>
<Timer name="RateGen_4x" period="100"/>
</Node>
<Hierarchy>
<Component name="RADAR">
<TaskContainment task="radar"/>
</Component>
<Component name="TACTICAL_STEERING">
<TaskContainment task="tactical_steering"/>
</Component>
<Component name="CURSOR_DEVICE">
<TaskContainment task="cursor_device"/>
```

```
</Component>
<Component name="SELECTED_POINT">
<TaskContainment task="selected_point"/>
</Component>
<Component name="TACTICAL_DISPLAY">
<TaskContainment task="tactical_display"/>
</Component>
<Component name="INS">
<TaskContainment task="ins"/>
</Component>
<Component name="GPS">
<TaskContainment task="gps"/>
</Component>
<Component name="AIRFRAME">
<TaskContainment task="airframe"/>
</Component>
<Component name="NAV_DISPLAY">
<TaskContainment task="nav_display"/>
</Component>
</Hierarchy>
<Dependency>
<TimerToTask timer="RateGen_1x" task="ins"/>
<TimerToTask timer="RateGen_1x" task="gps"/>
<TimerToTask timer="RateGen_2x" task="cursor_device"/>
<TimerToTask timer="RateGen_4x" task="radar"/>
<TaskToTask sourcetask="cursor_device" destinationtask="selected_point"/>
<TaskToTask sourcetask="radar" destinationtask="tactical_steering"/>
<TaskToChannel task="ins" channel="airframe_lc1"/>
<TaskToChannel task="gps" channel="airframe_lc2"/>
<TaskToChannel task="airframe" channel="nav_display_lc"/>
<TaskToChannel task="selected_point" channel="tactical_display_lc"/>
<TaskToChannel task="tactical_steering" channel="tactical_display_rc"/>
<TaskToChannel task="tactical_steering" channel="nav_display_rc"/>
<ChannelToTask channel="airframe_lc1" task="airframe"/>
<ChannelToTask channel="airframe_lc2" task="airframe"/>
<ChannelToTask channel="tactical_display_lc" task="tactical_display"/>
<ChannelToTask channel="nav_display_lc" task="nav_display"/>
<ChannelToTask channel="tactical_display_rc" task="tactical_display"/>
<ChannelToTask channel="nav_display_rc" task="nav_display"/>
</Dependency>
</DependencyAspect>
<PlatformAspect>
<Thread name="Thread_1" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="radar"/>
<TaskMapping task="tactical_steering"/>
<TimerMapping timer="RateGen_4x"/>
</Thread>
<Thread name="Thread_2" priority="2" queueingpolicy="FixedPriority">
<TaskMapping task="cursor_device"/>
<TaskMapping task="selected_point"/>
<TaskMapping task="tactical_display"/>
<TimerMapping timer="RateGen_2x"/>
</Thread>
<Thread name="Thread_3" priority="3" queueingpolicy="FixedPriority">
<TaskMapping task="ins"/>
<TaskMapping task="gps"/>
```

```
<TaskMapping task="airframe"/>
<TaskMapping task="nav_display"/>
<TimerMapping timer="RateGen_1x"/>
</Thread>
<Thread name="Channels" queueingpolicy="FixedPriority">
<ChannelMapping channel="airframe_lc1"/>
<ChannelMapping channel="airframe_lc2"/>
<ChannelMapping channel="tactical_display_lc"/>
<ChannelMapping channel="nav_display_lc"/>
<ChannelMapping channel="tactical_display_rc"/>
<ChannelMapping channel="nav_display_rc"/>
</Thread>
<CPU name="CPU_1" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_1"/>
</CPU>
<CPU name="CPU_2" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_2"/>
<ThreadMapping thread="Thread_3"/>
</CPU>
<CPU name="ChannelManager" schedulingpolicy="NonConcurrent">
<ThreadMapping thread="Channels"/>
</CPU>
</PlatformAspect>
</DRESystem>
```

10.5 Medium Distributed Non-preemptive ALDERIS XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../xsd/dream.xsd" name="medium_distributed_nonp"
version="0.2">
<DependencyAspect>
<Node>
<Task name="gps" deadline="22" subpriority="1" wcet="21"/>
<Task name="airframe" deadline="54" subpriority="2" wcet="53"/>
<Task name="pilot_waypoints" deadline="38" subpriority="3" wcet="37"/>
<Task name="routes" deadline="19" subpriority="4" wcet="18"/>
<Task name="navigator_navsteering_points" deadline="65" subpriority="5" wcet="32"/>
<Task name="nav_steering" deadline="50" subpriority="6" wcet="49"/>
<Task name="display_device" deadline="41" subpriority="7" wcet="26"/>
<Task name="af_monitor" deadline="34" subpriority="8" wcet="33"/>
<Task name="nav_display" deadline="74" subpriority="9" wcet="14"/>
<Task name="pilot_control" deadline="66" subpriority="10" wcet="23"/>
<Task name="tactical_steering" deadline="59" subpriority="11" wcet="58"/>
<Channel name="nav_steering_lc" buffersize="2"/>
<Channel name="routes_lc" buffersize="2"/>
<Channel name="tactical_steering_lc" buffersize="2"/>
<Channel name="nav_display_lc" buffersize="2"/>
<Channel name="af_monitor_lc" buffersize="2"/>
<Channel name="nav_steering_rc" bcdelay="1" buffersize="2" delay="2"/>
<Channel name="routes_rc" bcdelay="1" buffersize="2" delay="2"/>
<Channel name="tactical_steering_rc" bcdelay="1" buffersize="2" delay="3"/>
<Channel name="nav_display_rc1" bcdelay="1" buffersize="2" delay="3"/>
<Channel name="nav_display_rc2" bcdelay="1" buffersize="2" delay="3"/>
<Channel name="nav_display_rc3" bcdelay="1" buffersize="2" delay="2"/>
```

```
<Timer name="RateGen1x" period="1000"/>
<Timer name="RateGen2xa" period="500"/>
<Timer name="RateGen2xb" period="500"/>
<Timer name="RateGen10xa" period="100"/>
<Timer name="RateGen10xb" period="100"/>
</Node>
<Dependency>
<TimerToTask timer="RateGen1x" task="gps"/>
<TimerToTask timer="RateGen2xa" task="pilot_waypoints"/>
<TimerToTask timer="RateGen10xa" task="navigator_navsteering_points"/>
<TimerToTask timer="RateGen2xb" task="display_device"/>
<TimerToTask timer="RateGen10xb" task="pilot_control"/>
<TaskToTask sourcetask="gps" destinationtask="airframe"/>
<TaskToChannel task="pilot_waypoints" channel="routes_lc"/>
<TaskToChannel task="navigator_navsteering_points" channel="nav_steering_lc"/>
<TaskToChannel task="display_device" channel="nav_display_lc"/>
<TaskToChannel task="display_device" channel="af_monitor_lc"/>
<TaskToChannel task="pilot_control" channel="tactical_steering_lc"/>
<TaskToChannel task="airframe" channel="nav_steering_rc"/>
<TaskToChannel task="airframe" channel="routes_rc"/>
<TaskToChannel task="airframe" channel="tactical_steering_rc"/>
<TaskToChannel task="airframe" channel="nav_display_rc1"/>
<TaskToChannel task="tactical_steering" channel="nav_display_rc2"/>
<TaskToChannel task="nav_steering" channel="nav_display_rc3"/>
<ChannelToTask channel="nav_steering_lc" task="nav_steering"/>
<ChannelToTask channel="routes_lc" task="routes"/>
<ChannelToTask channel="tactical_steering_lc" task="tactical_steering"/>
<ChannelToTask channel="nav_display_lc" task="nav_display"/>
<ChannelToTask channel="af_monitor_lc" task="af_monitor"/>
<ChannelToTask channel="nav_steering_rc" task="nav_steering"/>
<ChannelToTask channel="routes_rc" task="routes"/>
<ChannelToTask channel="tactical_steering_rc" task="tactical_steering"/>
<ChannelToTask channel="nav_display_rc1" task="nav_display"/>
<ChannelToTask channel="nav_display_rc2" task="nav_display"/>
<ChannelToTask channel="nav_display_rc3" task="nav_display"/>
</Dependency>
</DependencyAspect>
<PlatformAspect>
<Thread name="Thread_1" queueingpolicy="FixedPriority">
<TaskMapping task="gps"/>
<TaskMapping task="airframe"/>
<TimerMapping timer="RateGen1x"/>
</Thread>
<Thread name="Thread_2" queueingpolicy="FixedPriority">
<TaskMapping task="pilot_waypoints"/>
<TaskMapping task="routes"/>
<TimerMapping timer="RateGen2xa"/>
</Thread>
<Thread name="Thread_3" queueingpolicy="FixedPriority">
<TaskMapping task="navigator_navsteering_points"/>
<TaskMapping task="nav_steering"/>
<TimerMapping timer="RateGen10xa"/>
</Thread>
<Thread name="Thread_4" queueingpolicy="FixedPriority">
<TaskMapping task="display_device"/>
<TaskMapping task="nav_display"/>
```

```
<TaskMapping task="af_monitor"/>
<TimerMapping timer="RateGen2xb"/>
</Thread>
<Thread name="Thread_5" queueingpolicy="FixedPriority">
<TaskMapping task="pilot_control"/>
<TaskMapping task="tactical_steering"/>
<TimerMapping timer="RateGen10xb"/>
</Thread>
<Thread name="Channels" queueingpolicy="FixedPriority">
<ChannelMapping channel="nav_steering_lc"/>
<ChannelMapping channel="routes_lc"/>
<ChannelMapping channel="tactical_steering_lc"/>
<ChannelMapping channel="nav_display_lc"/>
<ChannelMapping channel="af_monitor_lc"/>
<ChannelMapping channel="nav_steering_rc"/>
<ChannelMapping channel="routes_rc"/>
<ChannelMapping channel="tactical_steering_rc"/>
<ChannelMapping channel="nav_display_rc1"/>
<ChannelMapping channel="nav_display_rc2"/>
<ChannelMapping channel="nav_display_rc3"/>
</Thread>
<CPU name="CPU_1" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_1"/>
</CPU>
<CPU name="CPU_2" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_2"/>
</CPU>
<CPU name="CPU_3" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_3"/>
</CPU>
<CPU name="CPU_4" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_4"/>
</CPU>
<CPU name="CPU_5" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_5"/>
</CPU>
<CPU name="ChannelManager" schedulingpolicy="NonConcurrent">
<ThreadMapping thread="Channels"/>
</CPU>
</PlatformAspect>
</DRESystem>
```

10.6 Medium Distributed Preemptive ALDERIS XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../xsd/dream.xsd" name="medium_distributed" version="0.2">
<DependencyAspect>
<Node>
<Task name="gps" deadline="100" subpriority="1" wcet="21"/>
<Task name="airframe" deadline="100" subpriority="2" wcet="53"/>
<Task name="pilot_waypoints" deadline="300" subpriority="1" wcet="37"/>
<Task name="routes" deadline="250" subpriority="23" wcet="18"/>
<Task name="navigator_navsteering_points" deadline="100" subpriority="1" wcet="42"/>
<Task name="nav_steering" deadline="150" subpriority="2" wcet="69"/>
```

```
<Task name="display_device" deadline="250" subpriority="1" wcet="26"/>
<Task name="af_monitor" deadline="150" subpriority="2" wcet="33"/>
<Task name="nav_display" deadline="150" subpriority="3" wcet="14"/>
<Task name="pilot_control" deadline="80" subpriority="1" wcet="43"/>
<Task name="tactical_steering" deadline="100" subpriority="2" wcet="58"/>
<Channel name="nav_steering_lc" buffersize="2"/>
<Channel name="routes_lc" buffersize="2"/>
<Channel name="tactical_steering_lc" buffersize="2"/>
<Channel name="nav_display_lc" buffersize="2"/>
<Channel name="nav_steering_rc" buffersize="2" delay="2"/>
<Channel name="routes_rc" buffersize="2" delay="2"/>
<Channel name="tactical_steering_rc" buffersize="2" delay="3"/>
<Channel name="nav_display_rc" buffersize="2" delay="3"/>
<Timer name="RateGen_1x" period="1000"/>
<Timer name="RateGen_2xa" period="500"/>
<Timer name="RateGen_2xb" period="500"/>
<Timer name="RateGen_4xa" period="250"/>
<Timer name="RateGen_4xb" period="250"/>
</Node>
<Dependency>
<TimerToTask timer="RateGen_1x" task="gps"/>
<TimerToTask timer="RateGen_2xa" task="pilot_waypoints"/>
<TimerToTask timer="RateGen_4xa" task="navigator_navsteering_points"/>
<TimerToTask timer="RateGen_2xb" task="display_device"/>
<TimerToTask timer="RateGen_4xb" task="pilot_control"/>
<TaskToTask sourcetask="gps" destinationtask="airframe"/>
<TaskToTask sourcetask="nav_display" destinationtask="af_monitor"/>
<TaskToChannel task="pilot_waypoints" channel="routes_lc"/>
<TaskToChannel task="navigator_navsteering_points" channel="nav_steering_lc"/>
<TaskToChannel task="display_device" channel="nav_display_lc"/>
<TaskToChannel task="pilot_control" channel="tactical_steering_lc"/>
<TaskToChannel task="airframe" channel="nav_steering_rc"/>
<TaskToChannel task="airframe" channel="routes_rc"/>
<TaskToChannel task="airframe" channel="tactical_steering_rc"/>
<TaskToChannel task="airframe" channel="nav_display_rc"/>
<ChannelToTask channel="nav_steering_lc" task="nav_steering"/>
<ChannelToTask channel="routes_lc" task="routes"/>
<ChannelToTask channel="tactical_steering_lc" task="tactical_steering"/>
<ChannelToTask channel="nav_display_lc" task="nav_display"/>
<ChannelToTask channel="nav_steering_rc" task="nav_steering"/>
<ChannelToTask channel="routes_rc" task="routes"/>
<ChannelToTask channel="tactical_steering_rc" task="tactical_steering"/>
<ChannelToTask channel="nav_display_rc" task="nav_display"/>
</Dependency>
</DependencyAspect>
<PlatformAspect>
<Thread name="Thread_1" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="gps"/>
<TaskMapping task="airframe"/>
<TimerMapping timer="RateGen_1x"/>
</Thread>
<Thread name="Thread_2" priority="2" queueingpolicy="FixedPriority">
<TaskMapping task="pilot_waypoints"/>
<TaskMapping task="routes"/>
<TimerMapping timer="RateGen_2xa"/>
</Thread>
```

```
<Thread name="Thread_3" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="navigator_navsteering_points"/>
<TaskMapping task="nav_steering"/>
<TimerMapping timer="RateGen_4xa"/>
</Thread>
<Thread name="Thread_4" priority="2" queueingpolicy="FixedPriority">
<TaskMapping task="display_device"/>
<TaskMapping task="nav_display"/>
<TaskMapping task="af_monitor"/>
<TimerMapping timer="RateGen_2xb"/>
</Thread>
<Thread name="Thread_5" priority="1" queueingpolicy="FixedPriority">
<TaskMapping task="pilot_control"/>
<TaskMapping task="tactical_steering"/>
<TimerMapping timer="RateGen_4xb"/>
</Thread>
<Thread name="Channels" queueingpolicy="FixedPriority">
<ChannelMapping channel="nav_steering_lc"/>
<ChannelMapping channel="routes_lc"/>
<ChannelMapping channel="tactical_steering_lc"/>
<ChannelMapping channel="nav_display_lc"/>
<ChannelMapping channel="nav_steering_rc"/>
<ChannelMapping channel="routes_rc"/>
<ChannelMapping channel="tactical_steering_rc"/>
<ChannelMapping channel="nav_display_rc"/>
</Thread>
<CPU name="CPU_1" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_1"/>
</CPU>
<CPU name="CPU_2" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_2"/>
<ThreadMapping thread="Thread_3"/>
</CPU>
<CPU name="CPU_3" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_4"/>
<ThreadMapping thread="Thread_5"/>
</CPU>
<CPU name="ChannelManager" schedulingpolicy="NonConcurrent">
<ThreadMapping thread="Channels"/>
</CPU>
</PlatformAspect>
</DRESystem>
```

10.7 Large Distributed Non-preemptive ALDERIS XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DRESystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../xsd/dream.xsd" name="large_distributed_nonp"
version="0.1">
<DependencyAspect>
<Node>
<Task name="aths_device" deadline="1000" subpriority="0" wcet="2"/>
<Task name="aths_logical_device" deadline="1000" subpriority="1" wcet="2"/>
<Task name="aths_logical_display" deadline="1000" subpriority="2" wcet="2"/>
<Task name="dvms_device" deadline="1000" subpriority="3" wcet="2"/>
```

```
<Task name="dvms_logical_device" deadline="1000" subpriority="4" wcet="2"/>
<Task name="dvms_logical_display" deadline="1000" subpriority="5" wcet="2"/>
<Task name="radio1_device" deadline="1000" subpriority="6" wcet="2"/>
<Task name="radio1_logical_device" deadline="1000" subpriority="7" wcet="2"/>
<Task name="radio1_logical_display" deadline="1000" subpriority="8" wcet="2"/>
<Task name="radio2_device" deadline="1000" subpriority="9" wcet="2"/>
<Task name="radio2_logical_device" deadline="1000" subpriority="10" wcet="2"/>
<Task name="radio2_logical_display" deadline="1000" subpriority="11" wcet="2"/>
<Task name="miu_device" deadline="1000" subpriority="12" wcet="2"/>
<Task name="miu_logical_device" deadline="1000" subpriority="13" wcet="2"/>
<Task name="miu_logical_display" deadline="1000" subpriority="14" wcet="2"/>
<Task name="ems_device" deadline="1000" subpriority="15" wcet="2"/>
<Task name="ems_logical_device" deadline="1000" subpriority="16" wcet="2"/>
<Task name="ems_logical_display" deadline="1000" subpriority="17" wcet="2"/>
<Task name="radar_device" deadline="1000" subpriority="18" wcet="2"/>
<Task name="radar_logical_device" deadline="1000" subpriority="19" wcet="2"/>
<Task name="radar_logical_display" deadline="1000" subpriority="20" wcet="2"/>
<Task name="rwr_device" deadline="1000" subpriority="21" wcet="2"/>
<Task name="rwr_logical_device" deadline="1000" subpriority="22" wcet="2"/>
<Task name="rwr_logical_display" deadline="1000" subpriority="23" wcet="2"/>
<Task name="tacts_device" deadline="1000" subpriority="24" wcet="2"/>
<Task name="tacts_logical_device" deadline="1000" subpriority="25" wcet="2"/>
<Task name="tacts_logical_display" deadline="1000" subpriority="26" wcet="2"/>
<Task name="wmc_device" deadline="1000" subpriority="27" wcet="2"/>
<Task name="wmc_logical_device" deadline="1000" subpriority="28" wcet="2"/>
<Task name="wmc_logical_display" deadline="1000" subpriority="29" wcet="2"/>
<Task name="dmt_device" deadline="1000" subpriority="30" wcet="2"/>
<Task name="dmt_logical_device" deadline="1000" subpriority="31" wcet="2"/>
<Task name="dmt_logical_display" deadline="1000" subpriority="32" wcet="2"/>
<Task name="saahs_device" deadline="1000" subpriority="33" wcet="2"/>
<Task name="cnidc_device" deadline="1000" subpriority="34" wcet="2"/>
<Task name="af_state_logical_device" deadline="1000" subpriority="35" wcet="2"/>
<Task name="af_state_logical_display" deadline="1000" subpriority="36" wcet="2"/>
<Task name="device_5Hz" bcet="1" deadline="1000" subpriority="37" wcet="2"/>
<Task name="tactical_physical_device_5Hz" bcet="1" deadline="1000" subpriority="38" wcet="2"/>
<Task name="tactical_logical_device" bcet="1" deadline="1000" subpriority="39" wcet="2"/>
<Task name="tactical_mode" bcet="1" deadline="1000" subpriority="40" wcet="2"/>
<Task name="phase_manager" bcet="1" deadline="1000" subpriority="41" wcet="2"/>
<Task name="airframe" bcet="1" deadline="1000" subpriority="42" wcet="2"/>
<Task name="pilot_waypoints" bcet="1" deadline="1000" subpriority="43" wcet="2"/>
<Task name="routes" bcet="1" deadline="1000" subpriority="44" wcet="2"/>
<Task name="navigator_navsteering_points" bcet="1" deadline="1000" subpriority="45" wcet="2"/>
<Task name="nav_steering" bcet="1" deadline="1000" subpriority="46" wcet="2"/>
<Task name="display_device" bcet="1" deadline="1000" subpriority="47" wcet="2"/>
<Task name="nav_display" bcet="1" deadline="1000" subpriority="48" wcet="2"/>
<Task name="af_monitor" bcet="1" deadline="1000" subpriority="49" wcet="2"/>
<Task name="pilot_control" bcet="1" deadline="1000" subpriority="50" wcet="2"/>
<Task name="tactical_steering" bcet="1" deadline="1000" subpriority="51" wcet="2"/>
<Channel name="aths_1" buffersize="10"/>
<Channel name="aths_2" buffersize="10"/>
<Channel name="dvms_1" buffersize="10"/>
<Channel name="dvms_2" buffersize="10"/>
<Channel name="radio1_1" buffersize="10"/>
<Channel name="radio1_2" buffersize="10"/>
<Channel name="radio2_1" buffersize="10"/>
<Channel name="radio2_2" buffersize="10"/>
```

```
<Channel name="miu_1" buffersize="10"/>
<Channel name="miu_2" buffersize="10"/>
<Channel name="ems_1" buffersize="10"/>
<Channel name="ems_2" buffersize="10"/>
<Channel name="radar_1" buffersize="10"/>
<Channel name="radar_2" buffersize="10"/>
<Channel name="rwr_1" buffersize="10"/>
<Channel name="rwr_2" buffersize="10"/>
<Channel name="tacts_1" buffersize="10"/>
<Channel name="tacts_2" buffersize="10"/>
<Channel name="wmc_1" buffersize="10"/>
<Channel name="wmc_2" buffersize="10"/>
<Channel name="dmt_1" buffersize="10"/>
<Channel name="dmt_2" buffersize="10"/>
<Channel name="saahs_1" buffersize="10"/>
<Channel name="cnidc_1" buffersize="10"/>
<Channel name="af_state_logical_1" buffersize="10"/>
<Channel name="device_1" buffersize="10"/>
<Channel name="tactical_physical_1" buffersize="10"/>
<Channel name="tactical_logical_1" buffersize="10"/>
<Channel name="tactical_mode_1" buffersize="10"/>
<Channel name="tactical_logical_2" buffersize="10"/>
<Channel name="airframe_1" bcdelay="1" buffersize="10" delay="2"/>
/>
<Channel name="airframe_2" bcdelay="1" buffersize="10" delay="3"/>
/>
<Channel name="pilot_waypoints_1" buffersize="10"/>
<Channel name="navigator_navsteering_1" buffersize="10"/>
<Channel name="nav_steering_1" buffersize="10"/>
<Channel name="display_1" buffersize="10"/>
<Channel name="display_2" buffersize="10"/>
<Channel name="pilot_control_1" buffersize="10"/>
<Channel name="tactical_steering_1" buffersize="10"/>
<Timer name="RateGen_1Hz" period="1000"/>
<Timer name="RateGen_5Hz" period="200"/>
<Timer name="RateGen_10Hz" period="100"/>
<Timer name="RateGen_20Hz" period="50"/>
<Timer name="RateGen_40Hz" period="25"/>
</Node>
<Dependency>
<TimerToTask timer="RateGen_1Hz" task="saahs_device"/>
<TimerToTask timer="RateGen_1Hz" task="cnidc_device"/>
<TimerToTask timer="RateGen_1Hz" task="aths_device"/>
<TimerToTask timer="RateGen_1Hz" task="dvms_device"/>
<TimerToTask timer="RateGen_5Hz" task="device_5Hz"/>
<TimerToTask timer="RateGen_5Hz" task="pilot_waypoints"/>
<TimerToTask timer="RateGen_5Hz" task="navigator_navsteering_points"/>
<TimerToTask timer="RateGen_5Hz" task="radio1_device"/>
<TimerToTask timer="RateGen_5Hz" task="radio2_device"/>
<TimerToTask timer="RateGen_10Hz" task="miu_device"/>
<TimerToTask timer="RateGen_10Hz" task="display_device"/>
<TimerToTask timer="RateGen_10Hz" task="pilot_control"/>
<TimerToTask timer="RateGen_20Hz" task="ems_device"/>
<TimerToTask timer="RateGen_20Hz" task="radar_device"/>
<TimerToTask timer="RateGen_20Hz" task="rwr_device"/>
<TimerToTask timer="RateGen_20Hz" task="tacts_device"/>
```

```
<TimerToTask timer="RateGen_20Hz" task="wmc_device"/>
<TimerToTask timer="RateGen_40Hz" task="dmt_device"/>
<TaskToChannel task="aths_device" channel="aths_1"/>
<TaskToChannel task="aths_logical_device" channel="aths_2"/>
<TaskToChannel task="dvms_device" channel="dvms_1"/>
<TaskToChannel task="dvms_logical_device" channel="dvms_2"/>
<TaskToChannel task="radio1_device" channel="radio1_1"/>
<TaskToChannel task="radio1_logical_device" channel="radio1_2"/>
<TaskToChannel task="radio2_device" channel="radio2_1"/>
<TaskToChannel task="radio2_logical_device" channel="radio2_2"/>
<TaskToChannel task="miu_device" channel="miu_1"/>
<TaskToChannel task="miu_logical_device" channel="miu_2"/>
<TaskToChannel task="ems_device" channel="ems_1"/>
<TaskToChannel task="ems_logical_device" channel="ems_2"/>
<TaskToChannel task="radar_device" channel="radar_1"/>
<TaskToChannel task="radar_logical_device" channel="radar_2"/>
<TaskToChannel task="rwr_device" channel="rwr_1"/>
<TaskToChannel task="rwr_logical_device" channel="rwr_2"/>
<TaskToChannel task="tacts_device" channel="tacts_1"/>
<TaskToChannel task="tacts_logical_device" channel="tacts_2"/>
<TaskToChannel task="wmc_device" channel="wmc_1"/>
<TaskToChannel task="wmc_logical_device" channel="wmc_2"/>
<TaskToChannel task="dmt_device" channel="dmt_1"/>
<TaskToChannel task="dmt_logical_device" channel="dmt_2"/>
<TaskToChannel task="saahs_device" channel="saahs_1"/>
<TaskToChannel task="cnidc_device" channel="cnidc_1"/>
<TaskToChannel task="af_state_logical_device" channel="af_state_logical_1"/>
<TaskToChannel task="device_5Hz" channel="device_1"/>
<TaskToChannel task="tactical_physical_device_5Hz" channel="tactical_physical_1"/>
<TaskToChannel task="tactical_logical_device" channel="tactical_logical_1"/>
<TaskToChannel task="tactical_logical_device" channel="tactical_logical_2"/>
<TaskToChannel task="tactical_mode" channel="tactical_mode_1"/>
<TaskToChannel task="airframe" channel="airframe_1"/>
<TaskToChannel task="airframe" channel="airframe_2"/>
<TaskToChannel task="pilot_waypoints" channel="pilot_waypoints_1"/>
<TaskToChannel task="navigator_navsteering_points" channel="navigator_navsteering_1"/>
<TaskToChannel task="nav_steering" channel="nav_steering_1"/>
<TaskToChannel task="display_device" channel="display_1"/>
<TaskToChannel task="display_device" channel="display_2"/>
<TaskToChannel task="pilot_control" channel="pilot_control_1"/>
<TaskToChannel task="tactical_steering" channel="tactical_steering_1"/>
<ChannelToTask channel="aths_1" task="aths_logical_device"/>
<ChannelToTask channel="aths_2" task="aths_logical_display"/>
<ChannelToTask channel="dvms_1" task="dvms_logical_device"/>
<ChannelToTask channel="dvms_2" task="dvms_logical_display"/>
<ChannelToTask channel="radio1_1" task="radio1_logical_device"/>
<ChannelToTask channel="radio1_2" task="radio1_logical_display"/>
<ChannelToTask channel="radio2_1" task="radio2_logical_device"/>
<ChannelToTask channel="radio2_2" task="radio2_logical_display"/>
<ChannelToTask channel="miu_1" task="miu_logical_device"/>
<ChannelToTask channel="miu_2" task="miu_logical_display"/>
<ChannelToTask channel="ems_1" task="ems_logical_device"/>
<ChannelToTask channel="ems_2" task="ems_logical_display"/>
<ChannelToTask channel="radar_1" task="radar_logical_device"/>
<ChannelToTask channel="radar_2" task="radar_logical_display"/>
<ChannelToTask channel="rwr_1" task="rwr_logical_device"/>
```

```
<ChannelToTask channel="rwr_2" task="rwr_logical_display"/>
<ChannelToTask channel="tacts_1" task="tacts_logical_device"/>
<ChannelToTask channel="tacts_2" task="tacts_logical_display"/>
<ChannelToTask channel="wmc_1" task="wmc_logical_device"/>
<ChannelToTask channel="wmc_2" task="wmc_logical_display"/>
<ChannelToTask channel="dmt_1" task="dmt_logical_device"/>
<ChannelToTask channel="dmt_2" task="dmt_logical_display"/>
<ChannelToTask channel="saahs_1" task="af_state_logical_device"/>
<ChannelToTask channel="cnidc_1" task="af_state_logical_device"/>
<ChannelToTask channel="af_state_logical_1" task="af_state_logical_display"/>
<ChannelToTask channel="device_1" task="tactical_physical_device_5Hz"/>
<ChannelToTask channel="tactical_physical_1" task="tactical_logical_device"/>
<ChannelToTask channel="tactical_logical_1" task="tactical_mode"/>
<ChannelToTask channel="tactical_mode_1" task="phase_manager"/>
<ChannelToTask channel="tactical_logical_2" task="airframe"/>
<ChannelToTask channel="airframe_1" task="routes"/>
<ChannelToTask channel="airframe_2" task="nav_steering"/>
<ChannelToTask channel="pilot_waypoints_1" task="routes"/>
<ChannelToTask channel="navigator_navsteering_1" task="nav_steering"/>
<ChannelToTask channel="nav_steering_1" task="nav_display"/>
<ChannelToTask channel="display_1" task="nav_display"/>
<ChannelToTask channel="display_2" task="af_monitor"/>
<ChannelToTask channel="pilot_control_1" task="tactical_steering"/>
<ChannelToTask channel="tactical_steering_1" task="nav_display"/>
</Dependency>
</DependencyAspect>
<PlatformAspect>
<Thread name="Thread_1Hz" queueingpolicy="FixedPriority">
<TaskMapping task="aths_device"/>
<TaskMapping task="aths_logical_device"/>
<TaskMapping task="aths_logical_display"/>
<TaskMapping task="dvms_device"/>
<TaskMapping task="dvms_logical_device"/>
<TaskMapping task="dvms_logical_display"/>
<TaskMapping task="saahs_device"/>
<TaskMapping task="cnidc_device"/>
<TaskMapping task="af_state_logical_device"/>
<TaskMapping task="af_state_logical_display"/>
<TimerMapping timer="RateGen_1Hz"/>
</Thread>
<Thread name="Thread_5Hz" queueingpolicy="FixedPriority">
<TaskMapping task="radio1_device"/>
<TaskMapping task="radio1_logical_device"/>
<TaskMapping task="radio1_logical_display"/>
<TaskMapping task="radio2_device"/>
<TaskMapping task="radio2_logical_device"/>
<TaskMapping task="radio2_logical_display"/>
<TaskMapping task="device_5Hz"/>
<TaskMapping task="tactical_physical_device_5Hz"/>
<TaskMapping task="tactical_logical_device"/>
<TaskMapping task="tactical_mode"/>
<TaskMapping task="phase_manager"/>
<TaskMapping task="airframe"/>
<TaskMapping task="pilot_waypoints"/>
<TaskMapping task="routes"/>
<TaskMapping task="navigator_navsteering_points"/>
```

```
<TaskMapping task="nav_steering"/>
<TimerMapping timer="RateGen_5Hz"/>
</Thread>
<Thread name="Thread_10Hz" queueingpolicy="FixedPriority">
<TaskMapping task="miu_device"/>
<TaskMapping task="miu_logical_device"/>
<TaskMapping task="miu_logical_display"/>
<TaskMapping task="display_device"/>
<TaskMapping task="nav_display"/>
<TaskMapping task="af_monitor"/>
<TaskMapping task="pilot_control"/>
<TaskMapping task="tactical_steering"/>
<TimerMapping timer="RateGen_10Hz"/>
</Thread>
<Thread name="Thread_20Hz" queueingpolicy="FixedPriority">
<TaskMapping task="ems_device"/>
<TaskMapping task="ems_logical_device"/>
<TaskMapping task="ems_logical_display"/>
<TaskMapping task="radar_device"/>
<TaskMapping task="radar_logical_device"/>
<TaskMapping task="radar_logical_display"/>
<TaskMapping task="rwr_device"/>
<TaskMapping task="rwr_logical_device"/>
<TaskMapping task="rwr_logical_display"/>
<TaskMapping task="tacts_device"/>
<TaskMapping task="tacts_logical_device"/>
<TaskMapping task="tacts_logical_display"/>
<TaskMapping task="wmc_device"/>
<TaskMapping task="wmc_logical_device"/>
<TaskMapping task="wmc_logical_display"/>
<TimerMapping timer="RateGen_20Hz"/>
</Thread>
<Thread name="Thread_40Hz" queueingpolicy="FixedPriority">
<TaskMapping task="dmt_device"/>
<TaskMapping task="dmt_logical_device"/>
<TaskMapping task="dmt_logical_display"/>
<TimerMapping timer="RateGen_40Hz"/>
</Thread>
<Thread name="Channels" queueingpolicy="FixedPriority">
<ChannelMapping channel="aths_1"/>
<ChannelMapping channel="aths_2"/>
<ChannelMapping channel="dvms_1"/>
<ChannelMapping channel="dvms_2"/>
<ChannelMapping channel="radio1_1"/>
<ChannelMapping channel="radio1_2"/>
<ChannelMapping channel="radio2_1"/>
<ChannelMapping channel="radio2_2"/>
<ChannelMapping channel="miu_1"/>
<ChannelMapping channel="miu_2"/>
<ChannelMapping channel="ems_1"/>
<ChannelMapping channel="ems_2"/>
<ChannelMapping channel="radar_1"/>
<ChannelMapping channel="radar_2"/>
<ChannelMapping channel="rwr_1"/>
<ChannelMapping channel="rwr_2"/>
<ChannelMapping channel="tacts_1"/>
```

```
<ChannelMapping channel="tacts_2"/>
<ChannelMapping channel="wmc_1"/>
<ChannelMapping channel="wmc_2"/>
<ChannelMapping channel="dmt_1"/>
<ChannelMapping channel="dmt_2"/>
<ChannelMapping channel="saahs_1"/>
<ChannelMapping channel="cnidc_1"/>
<ChannelMapping channel="af_state_logical_1"/>
<ChannelMapping channel="device_1"/>
<ChannelMapping channel="tactical_physical_1"/>
<ChannelMapping channel="tactical_logical_1"/>
<ChannelMapping channel="tactical_mode_1"/>
<ChannelMapping channel="tactical_logical_2"/>
<ChannelMapping channel="airframe_1"/>
<ChannelMapping channel="airframe_2"/>
<ChannelMapping channel="pilot_waypoints_1"/>
<ChannelMapping channel="navigator_navsteering_1"/>
<ChannelMapping channel="nav_steering_1"/>
<ChannelMapping channel="display_1"/>
<ChannelMapping channel="display_2"/>
<ChannelMapping channel="pilot_control_1"/>
<ChannelMapping channel="tactical_steering_1"/>
</Thread>
<CPU name="CPU_1Hz" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_1Hz"/>
</CPU>
<CPU name="CPU_5Hz" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_5Hz"/>
</CPU>
<CPU name="CPU_10Hz" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_10Hz"/>
</CPU>
<CPU name="CPU_20Hz" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_20Hz"/>
</CPU>
<CPU name="CPU_40Hz" schedulingpolicy="FixedPriority">
<ThreadMapping thread="Thread_40Hz"/>
</CPU>
<CPU name="ChannelManager" schedulingpolicy="NonConcurrent">
<ThreadMapping thread="Channels"/>
</CPU>
</PlatformAspect>
</DRESystem>
```

10.8 Single CPU Non-preemptive Verimag IF Toolset Input

```
system dre;

signal event ();

/*
* Type declarations
*
*/
```

```
const pr=1;
const NTASKS=8;
type flags=array[NTASKS] of boolean;

const dl0=825;
const dl1=64;
const dl2=168;
const dl3=621;
const dl4=313;
const dl5=370;
const dl6=53;
const dl7=78;

const wcet0=31;
const wcet1=63;
const wcet2=38;
const wcet3=45;
const wcet4=33;
const wcet5=62;
const wcet6=52;
const wcet7=77;

const bcet0=23;
const bcet1=54;
const bcet2=21;
const bcet3=37;
const bcet4=21;
const bcet5=48;
const bcet6=32;
const bcet7=65;

/*
* Signalroutes
*
*
* Use signalroutes only if you have to model lossy channels
* or channels with delays.  The IF message passing builds on
* a buffered FIFO by default therefore it is suitable to
* model the real-time event channel mechanism as implemented
* in several middleware.  Try to avoid signalroutes as they
* tend to agressively increase the state space size.
*
*
* signalroute adc_dataavailable(1) #fifo #reliable #unicast #urgent;
* from task_adc to task_airframe
* with event;
*
* signalroute airframe_dataavailable1(1) #fifo #reliable #unicast #urgent;
* from task_airframe to task_tactical_steering
* with event;
*
* signalroute airframe_dataavailable2(1) #fifo #reliable #unicast #urgent;
* from task_airframe to task_nav_display
* with event;
*
* signalroute gps_dataavailable(1) #fifo #reliable #unicast #urgent;
```

```
* from task_gps to task_airframe
* with event;
*
* signalroute ins_dataavailable(1) #fifo #reliable #unicast #urgent;
* from task_ins to task_airframe
* with event;
*
* signalroute radar1_dataavailable(1) #fifo #reliable #unicast #urgent;
* from task_radar1 to task_airframe
* with event;
*
* signalroute radar2_dataavailable(1) #fifo #reliable #unicast #urgent;
* from task_radar2 to task_tactical_steering
* with event;
*
*/

/*
* Common variables
*
* We need a separate process to store them...
*/

process common(1);

var en flags public;
var exec flags public;

endprocess;

/*
* Timer - RateGen
*
*/

process timer_RateGen(1);

var ce clock;
const period_RateGen=1000;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen;
output event() to task_adc0;
output event() to task_gps0;
output event() to task_ins0;
output event() to task_radar10;
output event() to task_radar20;
set ce := 0;
nextstate timer;
```

```
endstate;

endprocess;

/*
* Task 0 - adc
*
*/

process task_adc(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[0] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[0];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet0 and ce >= bcet0;
task (common0).en[0] := false;
reset ce;
reset cd;
output event() /* via adc_dataavailable0 */ to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[0] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 1 - airframe
*
*/

process task_airframe(1);

var ce clock public;
var cd clock public;

state initial #start ;
```

```
deadline eager;
input event ();
set cd := 0;
task (common0).en[1] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[1];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet1 and ce >= bcet1;
task (common0).en[1] := false;
reset ce;
reset cd;
output event() /* via airframe_dataavailable10 */ to task_tactical_steering0;
output event() /* via airframe_dataavailable20 */ to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[1] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 2 - gps
*
*/

process task_gps(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[2] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[2];
set ce := 0;
nextstate run;
endstate;
```

```
state run;
deadline delayable;
when ce <= wcet2 and ce >= bcet2;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() /* via gps_dataavailable0 */ to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[2] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 3 - ins
*
*/

process task_ins(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[3] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[3];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet3 and ce >= bcet3;
task (common0).en[3] := false;
reset ce;
reset cd;
output event() /* via ins_dataavailable0 */ to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[3] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Task 4 - nav_display
*
*/

process task_nav_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[4] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[4];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet4 and ce >= bcet4;
task (common0).en[4] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[4] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 5 - radar1
*
*/

process task_radar1(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[5] := true;
```

```
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[5];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet5 and ce >= bcet5;
task (common0).en[5] := false;
reset ce;
reset cd;
output event() /* via radar1_dataavailable0 */ to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[5] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 6 - radar2
*
*/

process task_radar2(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[6] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[6];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet6 and ce >= bcet6;
task (common0).en[6] := false;
reset ce;
```

```
reset cd;
output event() /* via radar2_dataavailable0 */ to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[6] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 7 - tactical_steering
*
*/

process task_tactical_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[7] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[7];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet7 and ce >= bcet7;
task (common0).en[7] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[7] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/
```

```
process scheduler_CPU_1(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[0] and ((common0).en[1] = false) and ((common0).en[2] = false) and
((common0).en[3] = false) and ((common0).en[4] = false) and ((common0).en[5] = false)
and ((common0).en[6] = false) and ((common0).en[7] = false);
task (common0).exec[0] := true;
nextstate schedule_adc;

provided (common0).en[1] and ((common0).en[2] = false) and ((common0).en[6] = false) and
((common0).en[7] = false);
task (common0).exec[1] := true;
nextstate schedule_airframe;

provided (common0).en[2] and ((common0).en[6] = false) and ((common0).en[7] = false);
task (common0).exec[2] := true;
nextstate schedule_gps;

provided (common0).en[3] and ((common0).en[1] = false) and ((common0).en[2] = false) and
((common0).en[4] = false) and ((common0).en[5] = false) and ((common0).en[6] = false)
and ((common0).en[7] = false);
task (common0).exec[3] := true;
nextstate schedule_ins;

provided (common0).en[4] and ((common0).en[1] = false) and ((common0).en[2] = false) and
((common0).en[5] = false) and ((common0).en[6] = false) and ((common0).en[7] = false);
task (common0).exec[4] := true;
nextstate schedule_nav_display;

provided (common0).en[5] and ((common0).en[1] = false) and ((common0).en[2] = false) and
((common0).en[6] = false) and ((common0).en[7] = false);
task (common0).exec[5] := true;
nextstate schedule_radar1;

provided (common0).en[6];
task (common0).exec[6] := true;
nextstate schedule_radar2;

provided (common0).en[7] and ((common0).en[6] = false);
task (common0).exec[7] := true;
nextstate schedule_tactical_steering;

endstate;

state schedule_adc;
deadline eager;
provided (common0).en[1] or (common0).en[2] or (common0).en[3] or (common0).en[4] or (common0).en[5]
or (common0).en[6] or (common0).en[7];
```

```
when (task_adc0).ce = 0;
task (common0).exec[0] := false;
nextstate initial;
provided (common0).en[0] = false;
task (common0).exec[0] := false;
nextstate initial;

endstate;

state schedule_airframe;
deadline eager;
provided (common0).en[2] or (common0).en[6] or (common0).en[7];
when (task_airframe0).ce = 0;
task (common0).exec[1] := false;
nextstate initial;
provided (common0).en[1] = false;
task (common0).exec[1] := false;
nextstate initial;

endstate;

state schedule_gps;
deadline eager;
provided (common0).en[6] or (common0).en[7];
when (task_gps0).ce = 0;
task (common0).exec[2] := false;
nextstate initial;
provided (common0).en[2] = false;
task (common0).exec[2] := false;
nextstate initial;

endstate;

state schedule_ins;
deadline eager;
provided (common0).en[1] or (common0).en[2] or (common0).en[4] or (common0).en[5] or (common0).en[6]
or (common0).en[7];
when (task_ins0).ce = 0;
task (common0).exec[3] := false;
nextstate initial;
provided (common0).en[3] = false;
task (common0).exec[3] := false;
nextstate initial;

endstate;

state schedule_nav_display;
deadline eager;
provided (common0).en[1] or (common0).en[2] or (common0).en[5] or (common0).en[6] or (common0).en[7];
when (task_nav_display0).ce = 0;
task (common0).exec[4] := false;
nextstate initial;
provided (common0).en[4] = false;
task (common0).exec[4] := false;
nextstate initial;
```

```
endstate;

state schedule_radar1;
deadline eager;
provided (common0).en[1] or (common0).en[2] or (common0).en[6] or (common0).en[7];
when (task_radar10).ce = 0;
task (common0).exec[5] := false;
nextstate initial;
provided (common0).en[5] = false;
task (common0).exec[5] := false;
nextstate initial;

endstate;

state schedule_radar2;
provided (common0).en[6] = false;
task (common0).exec[6] := false;
nextstate initial;

endstate;

state schedule_tactical_steering;
deadline eager;
provided (common0).en[6];
when (task_tactical_steering0).ce = 0;
task (common0).exec[7] := false;
nextstate initial;
provided (common0).en[7] = false;
task (common0).exec[7] := false;
nextstate initial;

endstate;

endprocess;

endsystem;

/*
* Observer
*
*/

pure observer safety;

state idle #start ;
deadline eager;

provided (task_adc0) instate wait;
when (task_adc0).cd >= dl0;
nextstate error;
provided (task_adc0) instate run;
when (task_adc0).cd >= dl0;
nextstate error;

provided (task_airframe0) instate wait;
when (task_airframe0).cd >= dl1;
```

```
nextstate error;
provided (task_airframe0) instate run;
when (task_airframe0).cd >= dl1;
nextstate error;

provided (task_gps0) instate wait;
when (task_gps0).cd >= dl2;
nextstate error;
provided (task_gps0) instate run;
when (task_gps0).cd >= dl2;
nextstate error;

provided (task_ins0) instate wait;
when (task_ins0).cd >= dl3;
nextstate error;
provided (task_ins0) instate run;
when (task_ins0).cd >= dl3;
nextstate error;

provided (task_nav_display0) instate wait;
when (task_nav_display0).cd >= dl4;
nextstate error;
provided (task_nav_display0) instate run;
when (task_nav_display0).cd >= dl4;
nextstate error;

provided (task_radar10) instate wait;
when (task_radar10).cd >= dl5;
nextstate error;
provided (task_radar10) instate run;
when (task_radar10).cd >= dl5;
nextstate error;

provided (task_radar20) instate wait;
when (task_radar20).cd >= dl6;
nextstate error;
provided (task_radar20) instate run;
when (task_radar20).cd >= dl6;
nextstate error;

provided (task_tactical_steering0) instate wait;
when (task_tactical_steering0).cd >= dl7;
nextstate error;
provided (task_tactical_steering0) instate run;
when (task_tactical_steering0).cd >= dl7;
nextstate error;

endstate;

state error #error ;
endstate;

endobserver;
```

10.9 Multiple CPU Non-preemptive Verimag IF Toolset Input

```
system dre;

signal event ();

/*
* Type declarations
*
*/

const pr=1;
const NTASKS=8;
type flags=array[NTASKS] of boolean;

const dl0=177;
const dl1=64;
const dl2=39;
const dl3=146;
const dl4=318;
const dl5=101;
const dl6=53;
const dl7=111;

const wcet0=31;
const wcet1=63;
const wcet2=38;
const wcet3=45;
const wcet4=33;
const wcet5=62;
const wcet6=52;
const wcet7=77;

const bcet0=23;
const bcet1=54;
const bcet2=21;
const bcet3=37;
const bcet4=21;
const bcet5=48;
const bcet6=32;
const bcet7=65;

/*
* Signalroutes
*
*
* Use signalroutes only if you have to model lossy channels
* or channels with delays.  The IF message passing builds on
* a buffered FIFO by default therefore it is suitable to
* model the real-time event channel mechanism as implemented
* in several middleware.  Try to avoid signalroutes as they
* tend to agressively increase the state space size.
*
*
*/
```

```
signalroute adc_dataavailable(1) #fifo #reliable #unicast #delay[1,3];
from task_adc to task_airframe
with event;
/*
*/
signalroute airframe_dataavailable1(1) #fifo #reliable #unicast #delay[1,2];
from task_airframe to task_tactical_steering
with event;
/*
*/
signalroute airframe_dataavailable2(1) #fifo #reliable #unicast #delay[1,2];
from task_airframe to task_nav_display
with event;
/*
*/
signalroute gps_dataavailable(1) #fifo #reliable #unicast #delay[3,5];
from task_gps to task_airframe
with event;
/*
*/
signalroute ins_dataavailable(1) #fifo #reliable #unicast #delay[3,5];
from task_ins to task_airframe
with event;
/*
*/
signalroute radar1_dataavailable(1) #fifo #reliable #unicast #delay[0,2];
from task_radar1 to task_airframe
with event;
/*
* signalroute radar2_dataavailable(1) #fifo #reliable #unicast #urgent;
* from task_radar2 to task_tactical_steering
* with event;
*
*/

/*
* Common variables
*
* We need a separate process to store them...
*/

process common(1);

var en flags public;
var exec flags public;

endprocess;

/*
* Timer - RateGen
*
*/

process timer_RateGen(1);

var ce clock;
```

```
const period_RateGen=1000;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen;
output event() to task_adc0;
output event() to task_gps0;
output event() to task_ins0;
output event() to task_radar10;
output event() to task_radar20;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 0 - adc
*
*/

process task_adc(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[0] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[0];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet0 and ce >= bcet0;
task (common0).en[0] := false;
reset ce;
reset cd;
output event() via adc_dataavailable0 to task_airframe0;
nextstate initial;
deadline eager;
```

```
provided (common0).exec[0] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 2 - gps
*
*/

process task_gps(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[2] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[2];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet2 and ce >= bcet2;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() via gps_dataavailable0 to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[2] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

    /* * Task 3 - ins * */
    process task_ins(1);

    var ce clock public;
var cd clock public;

    state initial #start ;
deadline eager;
```

```
input event ();
set cd := 0;
task (common0).en[3] := true;
nextstate wait;
endstate;

    state wait;
deadline eager;
provided (common0).exec[3];
set ce := 0;
nextstate run;
endstate;

    state run;
deadline delayable;
when ce <= wcet3 and ce >= bcet3;
task (common0).en[3] := false;
reset ce;
reset cd;
output event() via ins_dataavailable0 to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[3] = false;
when ce = 0;
nextstate wait;
endstate;

    endprocess;

    /* * Task 5 - radar1 * */
    process task_radar1(1);

    var ce clock public;
var cd clock public;

    state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[5] := true;
nextstate wait;
endstate;

    state wait;
deadline eager;
provided (common0).exec[5];
set ce := 0;
nextstate run;
endstate;

    state run;
deadline delayable;
when ce <= wcet5 and ce >= bcet5;
task (common0).en[5] := false;
reset ce;
reset cd;
```

```
output event() via radar1_dataavailable0 to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[5] = false;
when ce = 0;
nextstate wait;
endstate;

    endprocess;

    /* * Task 1 - airframe * */
    process task_airframe(1);

    var ce clock public;
var cd clock public;

    state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[1] := true;
nextstate wait;
endstate;

    state wait;
deadline eager;
provided (common0).exec[1];
set ce := 0;
nextstate run;
endstate;

    state run;
deadline delayable;
when ce <= wcet1 and ce >= bcet1;
task (common0).en[1] := false;
reset ce;
reset cd;
output event() via airframe_dataavailable10 to task_tactical_steering0;
output event() via airframe_dataavailable20 to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[1] = false;
when ce = 0;
nextstate wait;
endstate;

    endprocess;

    /* * Task 4 - nav_display * */
    process task_nav_display(1);

    var ce clock public;
var cd clock public;

    state initial #start ;
deadline eager;
```

```
input event ();
set cd := 0;
task (common0).en[4] := true;
nextstate wait;
endstate;

    state wait;
deadline eager;
provided (common0).exec[4];
set ce := 0;
nextstate run;
endstate;

    state run;
deadline delayable;
when ce <= wcet4 and ce >= bcet4;
task (common0).en[4] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[4] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 6 - radar2
*
*/

process task_radar2(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[6] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[6];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet6 and ce >= bcet6;
```

```
task (common0).en[6] := false;
reset ce;
reset cd;
output event() /* via radar2_dataavailable0 */ to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[6] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 7 - tactical_steering
*
*/

process task_tactical_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[7] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[7];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet7 and ce >= bcet7;
task (common0).en[7] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[7] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
```

```
*/

process scheduler_CPU_1(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[0] and ((common0).en[2] = false) and ((common0).en[3] = false) and
((common0).en[5] = false);
task (common0).exec[0] := true;
nextstate schedule_adc;

provided (common0).en[2];
task (common0).exec[2] := true;
nextstate schedule_gps;

provided (common0).en[3] and ((common0).en[2] = false) and ((common0).en[5] = false);
task (common0).exec[3] := true;
nextstate schedule_ins;

provided (common0).en[5] and ((common0).en[2] = false);
task (common0).exec[5] := true;
nextstate schedule_radar1;

endstate;

state schedule_adc;
deadline eager;
provided (common0).en[2] or (common0).en[3] or (common0).en[5];
when (task_adc0).ce = 0;
task (common0).exec[0] := false;
nextstate initial;
provided (common0).en[0] = false;
task (common0).exec[0] := false;
nextstate initial;

endstate;

state schedule_gps;
provided (common0).en[2] = false;
task (common0).exec[2] := false;
nextstate initial;

endstate;

state schedule_ins;
deadline eager;
provided (common0).en[2] or (common0).en[5];
when (task_ins0).ce = 0;
task (common0).exec[3] := false;
```

```
nextstate initial;
provided (common0).en[3] = false;
task (common0).exec[3] := false;
nextstate initial;

    endstate;

state schedule_radar1;
deadline eager;
provided (common0).en[2];
when (task_radar10).ce = 0;
task (common0).exec[5] := false;
nextstate initial;
provided (common0).en[5] = false;
task (common0).exec[5] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_2(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[1];
task (common0).exec[1] := true;
nextstate schedule_airframe;

endstate;

state schedule_airframe;
provided (common0).en[1] = false;
task (common0).exec[1] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/
```

```
process scheduler_CPU_3(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[4] and ((common0).en[6] = false) and ((common0).en[7] = false);
task (common0).exec[4] := true;
nextstate schedule_nav_display;

provided (common0).en[6];
task (common0).exec[6] := true;
nextstate schedule_radar2;

provided (common0).en[7] and ((common0).en[6] = false);
task (common0).exec[7] := true;
nextstate schedule_tactical_steering;

endstate;

state schedule_nav_display;
deadline eager;
provided (common0).en[6] or (common0).en[7];
when (task_nav_display0).ce = 0;
task (common0).exec[4] := false;
nextstate initial;
provided (common0).en[4] = false;
task (common0).exec[4] := false;
nextstate initial;

endstate;

state schedule_radar2;
provided (common0).en[6] = false;
task (common0).exec[6] := false;
nextstate initial;

endstate;

state schedule_tactical_steering;
deadline eager;
provided (common0).en[6];
when (task_tactical_steering0).ce = 0;
task (common0).exec[7] := false;
nextstate initial;
provided (common0).en[7] = false;
task (common0).exec[7] := false;
nextstate initial;

endstate;
```

```
endprocess;

endsystem;

/*
* Observer
*
*/

pure observer safety;

state idle #start ;
deadline eager;

    provided (task_adc0) instate wait;
when (task_adc0).cd >= dl0;
nextstate error;
provided (task_adc0) instate run;
when (task_adc0).cd >= dl0;
nextstate error;

provided (task_airframe0) instate wait;
when (task_airframe0).cd >= dl1;
nextstate error;
provided (task_airframe0) instate run;
when (task_airframe0).cd >= dl1;
nextstate error;

provided (task_gps0) instate wait;
when (task_gps0).cd >= dl2;
nextstate error;
provided (task_gps0) instate run;
when (task_gps0).cd >= dl2;
nextstate error;

provided (task_ins0) instate wait;
when (task_ins0).cd >= dl3;
nextstate error;
provided (task_ins0) instate run;
when (task_ins0).cd >= dl3;
nextstate error;

provided (task_nav_display0) instate wait;
when (task_nav_display0).cd >= dl4;
nextstate error;
provided (task_nav_display0) instate run;
when (task_nav_display0).cd >= dl4;
nextstate error;

provided (task_radar10) instate wait;
when (task_radar10).cd >= dl5;
nextstate error;
provided (task_radar10) instate run;
when (task_radar10).cd >= dl5;
nextstate error;
```

```
provided (task_radar20) instate wait;
when (task_radar20).cd >= dl6;
nextstate error;
provided (task_radar20) instate run;
when (task_radar20).cd >= dl6;
nextstate error;

provided (task_tactical_steering0) instate wait;
when (task_tactical_steering0).cd >= dl7;
nextstate error;
provided (task_tactical_steering0) instate run;
when (task_tactical_steering0).cd >= dl7;
nextstate error;

endstate;

state error #error ;
endstate;

endobserver;
```

10.10 Small Distributed Non-preemptive Verimag IF Toolset Input

```
system dre;

signal event ();

/*
* Type declarations
*
*/

const pr=1;
const NTASKS=9;
type flags=array[NTASKS] of boolean;

const dl0=920;
const dl1=155;
const dl2=869;
const dl3=872;
const dl4=859;
const dl5=84;
const dl6=161;
const dl7=158;
const dl8=88;

const wcet0=80;
const wcet1=18;
const wcet2=29;
const wcet3=32;
const wcet4=19;
const wcet5=12;
const wcet6=24;
const wcet7=21;
```

```
const wcet8=16;

const bcet0=67;
const bcet1=13;
const bcet2=22;
const bcet3=27;
const bcet4=13;
const bcet5=8;
const bcet6=17;
const bcet7=9;
const bcet8=11;

/*
* Signalroutes
*
*
* Use signalroutes only if you have to model lossy channels
* or channels with delays.  The IF message passing builds on
* a buffered FIFO by default therefore it is suitable to
* model the real-time event channel mechanism as implemented
* in several middleware.  Try to avoid signalroutes as they
* tend to agressively increase the state space size.
*
*
* signalroute airframe_lc1(1) #fifo #reliable #unicast #urgent;
* from task_ins to task_airframe
* with event;
*
* signalroute airframe_lc2(1) #fifo #reliable #unicast #urgent;
* from task_gps to task_airframe
* with event;
*
* signalroute nav_display_lc(1) #fifo #reliable #unicast #urgent;
* from task_airframe to task_nav_display
* with event;
*
*/
signalroute nav_display_rc(1) #fifo #reliable #unicast #delay[0,2];
from task_tactical_steering to task_nav_display
with event;
/*
* signalroute tactical_display_lc(1) #fifo #reliable #unicast #urgent;
* from task_selected_point to task_tactical_display
* with event;
*
*/
signalroute tactical_display_rc(1) #fifo #reliable #unicast #delay[1,2];
from task_tactical_steering to task_tactical_display
with event;
/*
*/

/*
* Common variables
*
* We need a separate process to store them...
```

```
*/

process common(1);

var en flags public;
var exec flags public;

endprocess;

/*
* Timer - RateGen_4x
*
*/

process timer_RateGen_4x(1);

var ce clock;
const period_RateGen_4x=100;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_4x;
output event() to task_radar0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 5 - radar
*
*/

process task_radar(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[5] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[5];
```

```
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet5 and ce >= bcet5;
task (common0).en[5] := false;
reset ce;
reset cd;
output event() to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[5] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 8 - tactical_steering
*
*/

process task_tactical_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[8] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[8];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet8 and ce >= bcet8;
task (common0).en[8] := false;
reset ce;
reset cd;
output event() via nav_display_rc0 to task_nav_display0;
output event() via tactical_display_rc0 to task_tactical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[8] = false;
```

```
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_2x
*
*/

process timer_RateGen_2x(1);

var ce clock;
const period_RateGen_2x=200;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_2x;
output event() to task_cursor_device0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 1 - cursor_device
*
*/

process task_cursor_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[1] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[1];
set ce := 0;
nextstate run;
endstate;
```

```
state run;
deadline delayable;
when ce <= wcet1 and ce >= bcet1;
task (common0).en[1] := false;
reset ce;
reset cd;
output event() to task_selected_point0;
nextstate initial;
deadline eager;
provided (common0).exec[1] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 6 - selected_point
*
*/

process task_selected_point(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[6] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[6];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet6 and ce >= bcet6;
task (common0).en[6] := false;
reset ce;
reset cd;
output event() /* via tactical_display_lc0 */ to task_tactical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[6] = false;
when ce = 0;
nextstate wait;
endstate;
```

```
endprocess;

/*
* Task 7 - tactical_display
*
*/

process task_tactical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[7] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[7];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet7 and ce >= bcet7;
task (common0).en[7] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[7] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_1x
*
*/

process timer_RateGen_1x(1);

var ce clock;
const period_RateGen_1x=1000;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
```

```
endstate;

state timer;
deadline eager;
when ce = period_RateGen_1x;
output event() to task_gps0;
output event() to task_ins0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 0 - airframe
*
*/

process task_airframe(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[0] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[0];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet0 and ce >= bcet0;
task (common0).en[0] := false;
reset ce;
reset cd;
output event() /* via nav_display_lc0 */ to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[0] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 2 - gps
```

```
*
*/

process task_gps(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[2] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[2];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet2 and ce >= bcet2;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() /* via airframe_lc20 */ to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[2] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 3 - ins
*
*/

process task_ins(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[3] := true;
nextstate wait;
endstate;
```

```
state wait;
deadline eager;
provided (common0).exec[3];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet3 and ce >= bcet3;
task (common0).en[3] := false;
reset ce;
reset cd;
output event() /* via airframe_lc10 */ to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[3] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 4 - nav_display
*
*/

process task_nav_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[4] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[4];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet4 and ce >= bcet4;
task (common0).en[4] := false;
reset ce;
reset cd;
nextstate initial;
```

```
deadline eager;
provided (common0).exec[4] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_1(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[5];
task (common0).exec[5] := true;
nextstate schedule_radar;

    provided (common0).en[8] and ((common0).en[5] = false);
task (common0).exec[8] := true;
nextstate schedule_tactical_steering;

endstate;

state schedule_radar;
provided (common0).en[5] = false;
task (common0).exec[5] := false;
nextstate initial;

endstate;

state schedule_tactical_steering;
deadline eager;
provided (common0).en[5];
when (task_tactical_steering0).ce = 0;
task (common0).exec[8] := false;
nextstate initial;
provided (common0).en[8] = false;
task (common0).exec[8] := false;
nextstate initial;

endstate;

endprocess;

/*
```

```
* Fixed-priority scheduler
*
*/

process scheduler_CPU_2(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[1];
task (common0).exec[1] := true;
nextstate schedule_cursor_device;

provided (common0).en[6] and ((common0).en[1] = false);
task (common0).exec[6] := true;
nextstate schedule_selected_point;

provided (common0).en[7] and ((common0).en[1] = false) and ((common0).en[6] = false);
task (common0).exec[7] := true;
nextstate schedule_tactical_display;

endstate;

state schedule_cursor_device;
provided (common0).en[1] = false;
task (common0).exec[1] := false;
nextstate initial;

endstate;

state schedule_selected_point;
deadline eager;
provided (common0).en[1];
when (task_selected_point0).ce = 0;
task (common0).exec[6] := false;
nextstate initial;
provided (common0).en[6] = false;
task (common0).exec[6] := false;
nextstate initial;

endstate;

state schedule_tactical_display;
deadline eager;
/* * We have to hack the scheduler to compensate for * the race conditions caused by non-atomic
broadcasts...  * */ provided (common0).en[1] or (common0).en[6];
when (task_tactical_display0).ce = 0;
task (common0).exec[7] := false;
nextstate initial;
provided (common0).en[7] = false;
```

```
task (common0).exec[7] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_3(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[0] and ((common0).en[2] = false) and ((common0).en[3] = false);
task (common0).exec[0] := true;
nextstate schedule_airframe;

provided (common0).en[2] and ((common0).en[3] = false);
task (common0).exec[2] := true;
nextstate schedule_gps;

provided (common0).en[3];
task (common0).exec[3] := true;
nextstate schedule_ins;

provided (common0).en[4] and ((common0).en[0] = false) and ((common0).en[2] = false) and
((common0).en[3] = false);
task (common0).exec[4] := true;
nextstate schedule_nav_display;

endstate;

state schedule_airframe;
deadline eager;
provided (common0).en[2] or (common0).en[3];
when (task_airframe0).ce = 0;
task (common0).exec[0] := false;
nextstate initial;
provided (common0).en[0] = false;
task (common0).exec[0] := false;
nextstate initial;

endstate;

state schedule_gps;
deadline eager;
```

```
provided (common0).en[3];
when (task_gps0).ce = 0;
task (common0).exec[2] := false;
nextstate initial;
provided (common0).en[2] = false;
task (common0).exec[2] := false;
nextstate initial;

endstate;

state schedule_ins;
provided (common0).en[3] = false;
task (common0).exec[3] := false;
nextstate initial;

endstate;

state schedule_nav_display;
deadline eager;
provided (common0).en[0] or (common0).en[2] or (common0).en[3];
when (task_nav_display0).ce = 0;
task (common0).exec[4] := false;
nextstate initial;
provided (common0).en[4] = false;
task (common0).exec[4] := false;
nextstate initial;

endstate;

endprocess;

endsystem;

/*
* Observer
*
*/

pure observer safety;

state idle #start ;
deadline eager;

provided (task_airframe0) instate wait;
when (task_airframe0).cd >= dl0;
nextstate error;
provided (task_airframe0) instate run;
when (task_airframe0).cd >= dl0;
nextstate error;

provided (task_cursor_device0) instate wait;
when (task_cursor_device0).cd >= dl1;
nextstate error;
provided (task_cursor_device0) instate run;
when (task_cursor_device0).cd >= dl1;
nextstate error;
```

```
provided (task_gps0) instate wait;
when (task_gps0).cd >= dl2;
nextstate error;
provided (task_gps0) instate run;
when (task_gps0).cd >= dl2;
nextstate error;

provided (task_ins0) instate wait;
when (task_ins0).cd >= dl3;
nextstate error;
provided (task_ins0) instate run;
when (task_ins0).cd >= dl3;
nextstate error;

provided (task_nav_display0) instate wait;
when (task_nav_display0).cd >= dl4;
nextstate error;
provided (task_nav_display0) instate run;
when (task_nav_display0).cd >= dl4;
nextstate error;

provided (task_radar0) instate wait;
when (task_radar0).cd >= dl5;
nextstate error;
provided (task_radar0) instate run;
when (task_radar0).cd >= dl5;
nextstate error;

provided (task_selected_point0) instate wait;
when (task_selected_point0).cd >= dl6;
nextstate error;
provided (task_selected_point0) instate run;
when (task_selected_point0).cd >= dl6;
nextstate error;

provided (task_tactical_display0) instate wait;
when (task_tactical_display0).cd >= dl7;
nextstate error;
provided (task_tactical_display0) instate run;
when (task_tactical_display0).cd >= dl7;
nextstate error;

provided (task_tactical_steering0) instate wait;
when (task_tactical_steering0).cd >= dl8;
nextstate error;
provided (task_tactical_steering0) instate run;
when (task_tactical_steering0).cd >= dl8;
nextstate error;

endstate;

state error #error ;
endstate;

endobserver;
```

10.11 Small Distributed Preemptive Verimag IF Toolset Input

```
system dre;

signal event ();

/*
* Type declarations
*
*/

const pr=1;
const NTASKS=9;
type flags=array[NTASKS] of boolean;

const dl0=246;
const dl1=65;
const dl2=170;
const dl3=170;
const dl4=532;
const dl5=13;
const dl6=25;
const dl7=80;
const dl8=17;

const wcet0=80;
const wcet1=18;
const wcet2=29;
const wcet3=32;
const wcet4=19;
const wcet5=12;
const wcet6=24;
const wcet7=21;
const wcet8=16;

const bcet0=80;
const bcet1=18;
const bcet2=29;
const bcet3=32;
const bcet4=19;
const bcet5=12;
const bcet6=24;
const bcet7=21;
const bcet8=16;

/*
* Signalroutes
*
*
* Use signalroutes only if you have to model lossy channels
* or channels with delays.  The IF message passing builds on
* a buffered FIFO by default therefore it is suitable to
* model the real-time event channel mechanism as implemented
```

```
* in several middleware.  Try to avoid signalroutes as they
* tend to agressively increase the state space size.
*
*
* signalroute airframe_lc1(1) #fifo #reliable #unicast #urgent;
* from task_ins to task_airframe
* with event;
*
* signalroute airframe_lc2(1) #fifo #reliable #unicast #urgent;
* from task_gps to task_airframe
* with event;
*
* signalroute nav_display_lc(1) #fifo #reliable #unicast #urgent;
* from task_airframe to task_nav_display
* with event;
*
*/
signalroute nav_display_rc(1) #fifo #reliable #unicast #delay[2,2];
from task_tactical_steering to task_nav_display
with event;
/*
* signalroute tactical_display_lc(1) #fifo #reliable #unicast #urgent;
* from task_selected_point to task_tactical_display
* with event;
*
*/
signalroute tactical_display_rc(1) #fifo #reliable #unicast #delay[2,2];
from task_tactical_steering to task_tactical_display
with event;
/*
*/

/*
* Common variables
*
* We need a separate process to store them...
*/

process common(1);

var en flags public;
var exec flags public;

endprocess;

/*
* Timer - RateGen_4x
*
*/

process timer_RateGen_4x(1);

var ce clock;
const period_RateGen_4x=100;

state start #start ;
```

```
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_4x;
output event() to task_radar0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 5 - radar
*
*/

process task_radar(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[5] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[5];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet5 and ce >= bcet5;
task (common0).en[5] := false;
reset ce;
reset cd;
output event() to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[5] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Task 8 - tactical_steering
*
*/

process task_tactical_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[8] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[8];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet8 and ce >= bcet8;
task (common0).en[8] := false;
reset ce;
reset cd;
output event() via nav_display_rc0 to task_nav_display0;
output event() via tactical_display_rc0 to task_tactical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[8] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_2x
*
*/

process timer_RateGen_2x(1);

var ce clock;
const period_RateGen_2x=200;

    state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
```

```
endstate;

state timer;
deadline eager;
when ce = period_RateGen_2x;
output event() to task_cursor_device0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 1 - cursor_device
*
*/

process task_cursor_device(1);

var ce clock public;
var cd clock public;

    state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[1] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[1];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet1 and ce >= bcet1;
task (common0).en[1] := false;
reset ce;
reset cd;
output event() to task_selected_point0;
nextstate initial;
deadline eager;
provided (common0).exec[1] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 6 - selected_point
*
```

```
*/

process task_selected_point(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[6] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[6];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet6 and ce >= bcet6;
task (common0).en[6] := false;
reset ce;
reset cd;
output event() /* via tactical_display_lc0 */ to task_tactical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[6] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 7 - tactical_display
*
*/

process task_tactical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[7] := true;
nextstate wait;
endstate;
```

```
state wait;
deadline eager;
provided (common0).exec[7];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet7 and ce >= bcet7;
task (common0).en[7] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[7] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_1x
*
*/

process timer_RateGen_1x(1);

var ce clock;
const period_RateGen_1x=1000;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_1x;
output event() to task_gps0;
output event() to task_ins0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 0 - airframe
*
*/

process task_airframe(1);
```

```
var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[0] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[0];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet0 - et - pre;
task (common0).en[0] := false;
reset ce;
reset cd;
output event() /* via nav_display_lc0 */ to task_nav_display0;
nextstate initial;
deadline eager;
when ce >= wcet0 - et;
task (common0).en[0] := false;
reset ce;
reset cd;
output event() /* via nav_display_lc0 */ to task_nav_display0;
nextstate initial;
provided et + pr < wcet0;
when ce >= pr;
nextstate pass;
provided (common0).exec[0] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[0] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[0] = false and et > 0;
task pre := pre + pr;
nextstate wait;

endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
```

```
nextstate run;
endstate;

endprocess;

/*
* Task 2 - gps
*
*/

process task_gps(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[2] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[2];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet2 - et - pre;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() /* via airframe_lc20 */ to task_airframe0;
nextstate initial;
deadline eager;
when ce >= wcet2 - et;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() /* via airframe_lc20 */ to task_airframe0;
nextstate initial;
provided et + pr < wcet2;
when ce >= pr;
nextstate pass;
provided (common0).exec[2] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[2] = false and et = 0;
```

```
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[2] = false and et > 0;
task pre := pre + pr;
nextstate wait;

endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
* Task 3 - ins
*
*/

process task_ins(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[3] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[3];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet3 - et - pre;
task (common0).en[3] := false;
reset ce;
reset cd;
output event() /* via airframe_lc10 */ to task_airframe0;
nextstate initial;
deadline eager;
when ce >= wcet3 - et;
```

```
task (common0).en[3] := false;
reset ce;
reset cd;
output event() /* via airframe_lc10 */ to task_airframe0;
nextstate initial;
provided et + pr < wcet3;
when ce >= pr;
nextstate pass;
provided (common0).exec[3] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[3] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[3] = false and et > 0;
task pre := pre + pr;
nextstate wait;

endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
* Task 4 - nav_display
*
*/

process task_nav_display(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[4] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[4];
set ce := 0;
nextstate run;
```

```
endstate;

state run;
deadline delayable;
when ce >= bcet4 - et - pre;
task (common0).en[4] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
when ce >= wcet4 - et;
task (common0).en[4] := false;
reset ce;
reset cd;
nextstate initial;
provided et + pr < wcet4;
when ce >= pr;
nextstate pass;
provided (common0).exec[4] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[4] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[4] = false and et > 0;
task pre := pre + pr;
nextstate wait;

endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_1(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[5];
```

```
task (common0).exec[5] := true;
nextstate schedule_radar;

    provided (common0).en[8] and ((common0).en[5] = false);
task (common0).exec[8] := true;
nextstate schedule_tactical_steering;

    endstate;

state schedule_radar;
provided (common0).en[5] = false;
task (common0).exec[5] := false;
nextstate initial;

    endstate;

state schedule_tactical_steering;
deadline eager;
provided (common0).en[5];
when (task_tactical_steering0).ce = 0;
task (common0).exec[8] := false;
nextstate initial;
provided (common0).en[8] = false;
task (common0).exec[8] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_2(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[0] and ((common0).en[1] = false) and ((common0).en[2] = false) and
((common0).en[3] = false) and ((common0).en[6] = false) and ((common0).en[7] = false);
task (common0).exec[0] := true;
nextstate schedule_airframe;

provided (common0).en[1];
task (common0).exec[1] := true;
task (common0).exec[0] := false;
task (common0).exec[2] := false;
task (common0).exec[3] := false;
```

```
task (common0).exec[4] := false;
nextstate schedule_cursor_device;

provided (common0).en[2] and ((common0).en[1] = false) and ((common0).en[6] = false) and
((common0).en[7] = false);
task (common0).exec[2] := true;
nextstate schedule_gps;

provided (common0).en[3] and ((common0).en[1] = false) and ((common0).en[6] = false) and
((common0).en[7] = false);
task (common0).exec[3] := true;
nextstate schedule_ins;

provided (common0).en[4] and ((common0).en[0] = false) and ((common0).en[1] = false) and
((common0).en[2] = false) and ((common0).en[3] = false) and ((common0).en[6] = false)
and ((common0).en[7] = false);
task (common0).exec[4] := true;
nextstate schedule_nav_display;

provided (common0).en[6] and ((common0).en[1] = false);
task (common0).exec[6] := true;
task (common0).exec[0] := false;
task (common0).exec[2] := false;
task (common0).exec[3] := false;
task (common0).exec[4] := false;
nextstate schedule_selected_point;

provided (common0).en[7] and ((common0).en[1] = false) and ((common0).en[6] = false);
task (common0).exec[7] := true;
task (common0).exec[0] := false;
task (common0).exec[2] := false;
task (common0).exec[3] := false;
task (common0).exec[4] := false;
nextstate schedule_tactical_display;

endstate;

state schedule_airframe;
deadline eager;
provided ((common0).en[2] or (common0).en[3]) and (task_airframe0).et = 0;
when (task_airframe0).ce = 0;
task (common0).exec[0] := false;
nextstate initial;
/* Preemptions */
provided (common0).en[1] or (common0).en[6] or (common0).en[7];
task (common0).exec[0] := false;
nextstate initial;
provided (common0).en[0] = false;
task (common0).exec[0] := false;
nextstate initial;

endstate;

state schedule_cursor_device;
provided (common0).en[1] = false;
task (common0).exec[1] := false;
```

```
nextstate initial;

    endstate;

state schedule_gps;
/* Preemptions */
provided (common0).en[1] or (common0).en[6] or (common0).en[7];
task (common0).exec[2] := false;
nextstate initial;
provided (common0).en[2] = false;
task (common0).exec[2] := false;
nextstate initial;

endstate;

state schedule_ins;
/* Preemptions */
provided (common0).en[1] or (common0).en[6] or (common0).en[7];
task (common0).exec[3] := false;
nextstate initial;
provided (common0).en[3] = false;
task (common0).exec[3] := false;
nextstate initial;

endstate;

state schedule_nav_display;
deadline eager;
provided ((common0).en[0] or (common0).en[2] or (common0).en[3]) and (task_nav_display0).et
= 0;
when (task_nav_display0).ce = 0;
task (common0).exec[4] := false;
nextstate initial;
/* Preemptions */
provided (common0).en[1] or (common0).en[6] or (common0).en[7];
task (common0).exec[4] := false;
nextstate initial;
provided (common0).en[4] = false;
task (common0).exec[4] := false;
nextstate initial;

endstate;

state schedule_selected_point;
deadline eager;
provided (common0).en[1];
when (task_selected_point0).ce = 0;
task (common0).exec[6] := false;
nextstate initial;
provided (common0).en[6] = false;
task (common0).exec[6] := false;
nextstate initial;

endstate;

state schedule_tactical_display;
```

```
deadline eager;
provided (common0).en[1] or (common0).en[6];
when (task_tactical_display0).ce = 0;
task (common0).exec[7] := false;
nextstate initial;
provided (common0).en[7] = false;
task (common0).exec[7] := false;
nextstate initial;

endstate;

endprocess;

endsystem;

/*
* Observer
*
*/

pure observer safety;

state idle #start ;
deadline eager;

provided (task_airframe0) instate wait;
when (task_airframe0).cd >= dl0 - (task_airframe0).pre;
nextstate error;
provided (task_airframe0) instate run;
when (task_airframe0).cd >= dl0 - (task_airframe0).pre;
nextstate error;

provided (task_cursor_device0) instate wait;
when (task_cursor_device0).cd >= dl1;
nextstate error;
provided (task_cursor_device0) instate run;
when (task_cursor_device0).cd >= dl1;
nextstate error;

provided (task_gps0) instate wait;
when (task_gps0).cd >= dl2 - (task_gps0).pre;
nextstate error;
provided (task_gps0) instate run;
when (task_gps0).cd >= dl2 - (task_gps0).pre;
nextstate error;

provided (task_ins0) instate wait;
when (task_ins0).cd >= dl3 - (task_ins0).pre;
nextstate error;
provided (task_ins0) instate run;
when (task_ins0).cd >= dl3 - (task_ins0).pre;
nextstate error;

provided (task_nav_display0) instate wait;
when (task_nav_display0).cd >= dl4 - (task_nav_display0).pre;
nextstate error;
```

```
provided (task_nav_display0) instate run;
when (task_nav_display0).cd >= dl4 - (task_nav_display0).pre;
nextstate error;

provided (task_radar0) instate wait;
when (task_radar0).cd >= dl5;
nextstate error;
provided (task_radar0) instate run;
when (task_radar0).cd >= dl5;
nextstate error;

provided (task_selected_point0) instate wait;
when (task_selected_point0).cd >= dl6;
nextstate error;
provided (task_selected_point0) instate run;
when (task_selected_point0).cd >= dl6;
nextstate error;

provided (task_tactical_display0) instate wait;
when (task_tactical_display0).cd >= dl7;
nextstate error;
provided (task_tactical_display0) instate run;
when (task_tactical_display0).cd >= dl7;
nextstate error;

provided (task_tactical_steering0) instate wait;
when (task_tactical_steering0).cd >= dl8;
nextstate error;
provided (task_tactical_steering0) instate run;
when (task_tactical_steering0).cd >= dl8;
nextstate error;

endstate;

state error #error ;
endstate;

endobserver;
```

10.12 Medium Distributed Non-preemptive Verimag IF Toolset Input

```
system dre;

signal event ();

/*
* Type declarations
*
*/

const pr=1;
const NTASKS=11;
type flags=array[NTASKS] of boolean;
```

```
const dl0=34;
const dl1=54;
const dl2=41;
const dl3=22;
const dl4=74;
const dl5=50;
const dl6=65;
const dl7=66;
const dl8=38;
const dl9=19;
const dl10=59;

const wcet0=33;
const wcet1=53;
const wcet2=26;
const wcet3=21;
const wcet4=14;
const wcet5=49;
const wcet6=32;
const wcet7=23;
const wcet8=37;
const wcet9=18;
const wcet10=58;

const bcet0=33;
const bcet1=53;
const bcet2=26;
const bcet3=21;
const bcet4=14;
const bcet5=49;
const bcet6=32;
const bcet7=23;
const bcet8=37;
const bcet9=18;
const bcet10=58;

/*
* Signalroutes
*
*
* Use signalroutes only if you have to model lossy channels
* or channels with delays.  The IF message passing builds on
* a buffered FIFO by default therefore it is suitable to
* model the real-time event channel mechanism as implemented
* in several middleware.  Try to avoid signalroutes as they
* tend to agressively increase the state space size.
*
*
* signalroute af_monitor_lc(1) #fifo #reliable #unicast #urgent;
* from task_display_device to task_af_monitor
* with event;
*
* signalroute nav_display_lc(1) #fifo #reliable #unicast #urgent;
* from task_display_device to task_nav_display
* with event;
*
```

```
*/
signalroute nav_display_rc1(1) #fifo #reliable #unicast #delay[1,3];
from task_airframe to task_nav_display
with event;
/*
*/
signalroute nav_display_rc2(1) #fifo #reliable #unicast #delay[1,3];
from task_tactical_steering to task_nav_display
with event;
/*
*/
signalroute nav_display_rc3(1) #fifo #reliable #unicast #delay[1,2];
from task_nav_steering to task_nav_display
with event;
/*
* signalroute nav_steering_lc(1) #fifo #reliable #unicast #urgent;
* from task_navigator_navsteering_points to task_nav_steering
* with event;
*
*/
signalroute nav_steering_rc(1) #fifo #reliable #unicast #delay[1,2];
from task_airframe to task_nav_steering
with event;
/*
* signalroute routes_lc(1) #fifo #reliable #unicast #urgent;
* from task_pilot_waypoints to task_routes
* with event;
*
*/
signalroute routes_rc(1) #fifo #reliable #unicast #delay[1,2];
from task_airframe to task_routes
with event;
/*
* signalroute tactical_steering_lc(1) #fifo #reliable #unicast #urgent;
* from task_pilot_control to task_tactical_steering
* with event;
*
*/
signalroute tactical_steering_rc(1) #fifo #reliable #unicast #delay[1,3];
from task_airframe to task_tactical_steering
with event;
/*
*/

/*
* Common variables
*
* We need a separate process to store them...
*/

process common(1);

var en flags public;
var exec flags public;

endprocess;
```

```
/*
* Timer - RateGen1x
*
*/

process timer_RateGen1x(1);

var ce clock;
const period_RateGen1x=1000;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen1x;
output event() to task_gps0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 1 - airframe
*
*/

process task_airframe(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[1] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[1];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet1 and ce >= bcet1;
task (common0).en[1] := false;
```

```
reset ce;
reset cd;
output event() via nav_display_rc10 to task_nav_display0;
output event() via nav_steering_rc0 to task_nav_steering0;
output event() via routes_rc0 to task_routes0;
output event() via tactical_steering_rc0 to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[1] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 3 - gps
*
*/

process task_gps(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[3] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[3];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet3 and ce >= bcet3;
task (common0).en[3] := false;
reset ce;
reset cd;
output event() to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[3] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Timer - RateGen2xa
*
*/

process timer_RateGen2xa(1);

var ce clock;
const period_RateGen2xa=500;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen2xa;
output event() to task_pilot_waypoints0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 8 - pilot_waypoints
*
*/

process task_pilot_waypoints(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[8] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[8];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet8 and ce >= bcet8;
task (common0).en[8] := false;
reset ce;
```

```
reset cd;
output event() /* via routes_lc0 */ to task_routes0;
nextstate initial;
deadline eager;
provided (common0).exec[8] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 9 - routes
*
*/

process task_routes(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[9] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[9];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet9 and ce >= bcet9;
task (common0).en[9] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[9] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen10xa
*
*/
```

```
process timer_RateGen10xa(1);

var ce clock;
const period_RateGen10xa=100;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen10xa;
output event() to task_navigator_navsteering_points0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 5 - nav_steering
*
*/

process task_nav_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[5] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[5];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet5 and ce >= bcet5;
task (common0).en[5] := false;
reset ce;
reset cd;
output event() via nav_display_rc30 to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[5] = false;
```

```
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 6 - navigator_navsteering_points
*
*/

process task_navigator_navsteering_points(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[6] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[6];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet6 and ce >= bcet6;
task (common0).en[6] := false;
reset ce;
reset cd;
output event() /* via nav_steering_lc0 */ to task_nav_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[6] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen2xb
*
*/

process timer_RateGen2xb(1);

var ce clock;
const period_RateGen2xb=500;
```

```
state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen2xb;
output event() to task_display_device0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 0 - af_monitor
*
*/

process task_af_monitor(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[0] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[0];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet0 and ce >= bcet0;
task (common0).en[0] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[0] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Task 2 - display_device
*
*/

process task_display_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[2] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[2];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet2 and ce >= bcet2;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() /* via af_monitor_lc0 */ to task_af_monitor0;
output event() /* via nav_display_lc0 */ to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[2] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 4 - nav_display
*
*/

process task_nav_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
```

```
set cd := 0;
task (common0).en[4] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[4];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet4 and ce >= bcet4;
task (common0).en[4] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[4] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen10xb
*
*/

process timer_RateGen10xb(1);

var ce clock;
const period_RateGen10xb=100;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen10xb;
output event() to task_pilot_control0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 7 - pilot_control
*
```

```
*/

process task_pilot_control(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[7] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[7];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet7 and ce >= bcet7;
task (common0).en[7] := false;
reset ce;
reset cd;
output event() /* via tactical_steering_lc0 */ to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[7] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 10 - tactical_steering
*
*/

process task_tactical_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[10] := true;
nextstate wait;
endstate;
```

```
state wait;
deadline eager;
provided (common0).exec[10];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet10 and ce >= bcet10;
task (common0).en[10] := false;
reset ce;
reset cd;
output event() via nav_display_rc20 to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[10] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_1(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[1] and ((common0).en[3] = false);
task (common0).exec[1] := true;
nextstate schedule_airframe;

provided (common0).en[3];
task (common0).exec[3] := true;
nextstate schedule_gps;

endstate;

state schedule_airframe;
deadline eager;
provided (common0).en[3];
when (task_airframe0).ce = 0;
task (common0).exec[1] := false;
nextstate initial;
provided (common0).en[1] = false;
```

```
task (common0).exec[1] := false;
nextstate initial;

endstate;

state schedule_gps;
provided (common0).en[3] = false;
task (common0).exec[3] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_2(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[8];
task (common0).exec[8] := true;
nextstate schedule_pilot_waypoints;

provided (common0).en[9] and ((common0).en[8] = false);
task (common0).exec[9] := true;
nextstate schedule_routes;

endstate;

state schedule_pilot_waypoints;
provided (common0).en[8] = false;
task (common0).exec[8] := false;
nextstate initial;

    endstate;

state schedule_routes;
deadline eager;
provided (common0).en[8];
when (task_routes0).ce = 0;
task (common0).exec[9] := false;
nextstate initial;
provided (common0).en[9] = false;
task (common0).exec[9] := false;
nextstate initial;
```

```
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_3(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[5] and ((common0).en[6] = false);
task (common0).exec[5] := true;
nextstate schedule_nav_steering;

provided (common0).en[6];
task (common0).exec[6] := true;
nextstate schedule_navigator_navsteering_points;

    endstate;

state schedule_nav_steering;
deadline eager;
provided (common0).en[6];
when (task_nav_steering0).ce = 0;
task (common0).exec[5] := false;
nextstate initial;
provided (common0).en[5] = false;
task (common0).exec[5] := false;
nextstate initial;

endstate;

state schedule_navigator_navsteering_points;
provided (common0).en[6] = false;
task (common0).exec[6] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/
```

```
process scheduler_CPU_4(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[0] and ((common0).en[2] = false);
task (common0).exec[0] := true;
nextstate schedule_af_monitor;

provided (common0).en[2];
task (common0).exec[2] := true;
nextstate schedule_display_device;

provided (common0).en[4] and ((common0).en[0] = false) and ((common0).en[2] = false);
task (common0).exec[4] := true;
nextstate schedule_nav_display;

endstate;

state schedule_af_monitor;
deadline eager;
provided (common0).en[2];
when (task_af_monitor0).ce = 0;
task (common0).exec[0] := false;
nextstate initial;
provided (common0).en[0] = false;
task (common0).exec[0] := false;
nextstate initial;

endstate;

state schedule_display_device;
provided (common0).en[2] = false;
task (common0).exec[2] := false;
nextstate initial;

endstate;

state schedule_nav_display;
deadline eager;
provided (common0).en[0] or (common0).en[2];
when (task_nav_display0).ce = 0;
task (common0).exec[4] := false;
nextstate initial;
provided (common0).en[4] = false;
task (common0).exec[4] := false;
nextstate initial;

endstate;
```

```
endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_5(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[7];
task (common0).exec[7] := true;
nextstate schedule_pilot_control;

provided (common0).en[10] and ((common0).en[7] = false);
task (common0).exec[10] := true;
nextstate schedule_tactical_steering;

endstate;

state schedule_pilot_control;
provided (common0).en[7] = false;
task (common0).exec[7] := false;
nextstate initial;

endstate;

state schedule_tactical_steering;
deadline eager;
provided (common0).en[7];
when (task_tactical_steering0).ce = 0;
task (common0).exec[10] := false;
nextstate initial;
provided (common0).en[10] = false;
task (common0).exec[10] := false;
nextstate initial;

endstate;

endprocess;

endsystem;

/*
* Observer
*
*/
```

```
pure observer safety;

state idle #start ;
deadline eager;

provided (task_af_monitor0) instate wait;
when (task_af_monitor0).cd >= dl0;
nextstate error;
provided (task_af_monitor0) instate run;
when (task_af_monitor0).cd >= dl0;
nextstate error;

provided (task_airframe0) instate wait;
when (task_airframe0).cd >= dl1;
nextstate error;
provided (task_airframe0) instate run;
when (task_airframe0).cd >= dl1;
nextstate error;

provided (task_display_device0) instate wait;
when (task_display_device0).cd >= dl2;
nextstate error;
provided (task_display_device0) instate run;
when (task_display_device0).cd >= dl2;
nextstate error;

provided (task_gps0) instate wait;
when (task_gps0).cd >= dl3;
nextstate error;
provided (task_gps0) instate run;
when (task_gps0).cd >= dl3;
nextstate error;

provided (task_nav_display0) instate wait;
when (task_nav_display0).cd >= dl4;
nextstate error;
provided (task_nav_display0) instate run;
when (task_nav_display0).cd >= dl4;
nextstate error;

provided (task_nav_steering0) instate wait;
when (task_nav_steering0).cd >= dl5;
nextstate error;
provided (task_nav_steering0) instate run;
when (task_nav_steering0).cd >= dl5;
nextstate error;

provided (task_navigator_navsteering_points0) instate wait;
when (task_navigator_navsteering_points0).cd >= dl6;
nextstate error;
provided (task_navigator_navsteering_points0) instate run;
when (task_navigator_navsteering_points0).cd >= dl6;
nextstate error;

provided (task_pilot_control0) instate wait;
```

```
when (task_pilot_control0).cd >= dl7;
nextstate error;
provided (task_pilot_control0) instate run;
when (task_pilot_control0).cd >= dl7;
nextstate error;

provided (task_pilot_waypoints0) instate wait;
when (task_pilot_waypoints0).cd >= dl8;
nextstate error;
provided (task_pilot_waypoints0) instate run;
when (task_pilot_waypoints0).cd >= dl8;
nextstate error;

provided (task_routes0) instate wait;
when (task_routes0).cd >= dl9;
nextstate error;
provided (task_routes0) instate run;
when (task_routes0).cd >= dl9;
nextstate error;

provided (task_tactical_steering0) instate wait;
when (task_tactical_steering0).cd >= dl10;
nextstate error;
provided (task_tactical_steering0) instate run;
when (task_tactical_steering0).cd >= dl10;
nextstate error;

endstate;

state error #error ;
endstate;

endobserver;
```

10.13 Medium Distributed Preemptive Verimag IF Toolset Input

```
system dre;

signal event ();

/*
* Type declarations
*
*/

const pr=1;
const NTASKS=11;
type flags=array[NTASKS] of boolean;

const dl0=150;
const dl1=100;
const dl2=250;
const dl3=100;
const dl4=150;
```

```
const dl5=150;
const dl6=100;
const dl7=80;
const dl8=300;
const dl9=250;
const dl10=100;

const wcet0=33;
const wcet1=53;
const wcet2=26;
const wcet3=21;
const wcet4=14;
const wcet5=69;
const wcet6=42;
const wcet7=43;
const wcet8=37;
const wcet9=18;
const wcet10=58;

const bcet0=33;
const bcet1=53;
const bcet2=26;
const bcet3=21;
const bcet4=14;
const bcet5=69;
const bcet6=42;
const bcet7=43;
const bcet8=37;
const bcet9=18;
const bcet10=58;

/*
* Signalroutes
*
*
* Use signalroutes only if you have to model lossy channels
* or channels with delays.  The IF message passing builds on
* a buffered FIFO by default therefore it is suitable to
* model the real-time event channel mechanism as implemented
* in several middleware.  Try to avoid signalroutes as they
* tend to agressively increase the state space size.
*
*
* signalroute nav_display_lc(1) #fifo #reliable #unicast #urgent;
* from task_display_device to task_nav_display
* with event;
*
*/
signalroute nav_display_rc(1) #fifo #reliable #unicast #delay[3,3];
from task_airframe to task_nav_display
with event;
/*
* signalroute nav_steering_lc(1) #fifo #reliable #unicast #urgent;
* from task_navigator_navsteering_points to task_nav_steering
* with event;
*
```

```
*/
signalroute nav_steering_rc(1) #fifo #reliable #unicast #delay[2,2];
from task_airframe to task_nav_steering
with event;
/*
* signalroute routes_lc(1) #fifo #reliable #unicast #urgent;
* from task_pilot_waypoints to task_routes
* with event;
*
*/
signalroute routes_rc(1) #fifo #reliable #unicast #delay[2,2];
from task_airframe to task_routes
with event;
/*
* signalroute tactical_steering_lc(1) #fifo #reliable #unicast #urgent;
* from task_pilot_control to task_tactical_steering
* with event;
*
*/
signalroute tactical_steering_rc(1) #fifo #reliable #unicast #delay[3,3];
from task_airframe to task_tactical_steering
with event;
/*
*/

/*
* Common variables
*
* We need a separate process to store them...
*/

process common(1);

var en flags public;
var exec flags public;

endprocess;

/*
* Timer - RateGen_1x
*
*/

process timer_RateGen_1x(1);

var ce clock;
const period_RateGen_1x=1000;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
```

```
when ce = period_RateGen_1x;
output event() to task_gps0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 1 - airframe
*
*/

process task_airframe(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[1] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[1];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet1 and ce >= bcet1;
task (common0).en[1] := false;
reset ce;
reset cd;
output event() via nav_display_rc0 to task_nav_display0;
output event() via nav_steering_rc0 to task_nav_steering0;
output event() via routes_rc0 to task_routes0;
output event() via tactical_steering_rc0 to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[1] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 3 - gps
*
*/
```

```
process task_gps(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[3] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[3];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet3 and ce >= bcet3;
task (common0).en[3] := false;
reset ce;
reset cd;
output event() to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[3] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_4xa
*
*/

process timer_RateGen_4xa(1);

var ce clock;
const period_RateGen_4xa=250;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_4xa;
```

```
output event() to task_navigator_navsteering_points0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 5 - nav_steering
*
*/

process task_nav_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[5] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[5];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet5 and ce >= bcet5;
task (common0).en[5] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[5] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 6 - navigator_navsteering_points
*
*/

process task_navigator_navsteering_points(1);

var ce clock public;
var cd clock public;
```

```
state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[6] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[6];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet6 and ce >= bcet6;
task (common0).en[6] := false;
reset ce;
reset cd;
output event() /* via nav_steering_lc0 */ to task_nav_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[6] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_2xa
*
*/

process timer_RateGen_2xa(1);

var ce clock;
const period_RateGen_2xa=500;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_2xa;
output event() to task_pilot_waypoints0;
set ce := 0;
nextstate timer;
endstate;
```

```
endprocess;

/*
* Task 8 - pilot_waypoints
*
*/

process task_pilot_waypoints(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[8] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[8];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet8 - et - pre;
task (common0).en[8] := false;
reset ce;
reset cd;
output event() /* via routes_lc0 */ to task_routes0;
nextstate initial;
deadline eager;
when ce >= wcet8 - et;
task (common0).en[8] := false;
reset ce;
reset cd;
output event() /* via routes_lc0 */ to task_routes0;
nextstate initial;
provided et + pr < wcet8;
when ce >= pr;
nextstate pass;
provided (common0).exec[8] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[8] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
```

```
provided (common0).exec[8] = false and et > 0;
task pre := pre + pr;
nextstate wait;

    endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
* Task 9 - routes
*
*/

process task_routes(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[9] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[9];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet9 - et - pre;
task (common0).en[9] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
when ce >= wcet9 - et;
task (common0).en[9] := false;
reset ce;
reset cd;
nextstate initial;
```

```
provided et + pr < wcet9;
when ce >= pr;
nextstate pass;
provided (common0).exec[9] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[9] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[9] = false and et > 0;
task pre := pre + pr;
nextstate wait;

    endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
* Timer - RateGen_4xb
*
*/

process timer_RateGen_4xb(1);

var ce clock;
const period_RateGen_4xb=250;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_4xb;
output event() to task_pilot_control0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 7 - pilot_control
*
*/

process task_pilot_control(1);
```

```
var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[7] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[7];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet7 and ce >= bcet7;
task (common0).en[7] := false;
reset ce;
reset cd;
output event() /* via tactical_steering_lc0 */ to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[7] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 10 - tactical_steering
*
*/

process task_tactical_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[10] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[10];
```

```
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet10 and ce >= bcet10;
task (common0).en[10] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[10] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_2xb
*
*/

process timer_RateGen_2xb(1);

var ce clock;
const period_RateGen_2xb=500;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_2xb;
output event() to task_display_device0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 0 - af_monitor
*
*/

process task_af_monitor(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;
```

```
state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[0] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[0];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet0 - et - pre;
task (common0).en[0] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
when ce >= wcet0 - et;
task (common0).en[0] := false;
reset ce;
reset cd;
nextstate initial;
provided et + pr < wcet0;
when ce >= pr;
nextstate pass;
provided (common0).exec[0] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[0] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[0] = false and et > 0;
task pre := pre + pr;
nextstate wait;

endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
```

```
* Task 2 - display_device
*
*/

process task_display_device(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[2] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[2];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet2 - et - pre;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() /* via nav_display_lc0 */ to task_nav_display0;
nextstate initial;
deadline eager;
when ce >= wcet2 - et;
task (common0).en[2] := false;
reset ce;
reset cd;
output event() /* via nav_display_lc0 */ to task_nav_display0;
nextstate initial;
provided et + pr < wcet2;
when ce >= pr;
nextstate pass;
provided (common0).exec[2] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[2] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[2] = false and et > 0;
task pre := pre + pr;
nextstate wait;
```

```
    endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
* Task 4 - nav_display
*
*/

process task_nav_display(1);

var ce clock public;
var cd clock public;
var et integer public;
var pre integer public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task et := 0;
task pre := 0;
task (common0).en[4] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[4];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce >= bcet4 - et - pre;
task (common0).en[4] := false;
reset ce;
reset cd;
output event() to task_af_monitor0;
nextstate initial;
deadline eager;
when ce >= wcet4 - et;
task (common0).en[4] := false;
reset ce;
reset cd;
output event() to task_af_monitor0;
nextstate initial;
provided et + pr < wcet4;
```

```
when ce >= pr;
nextstate pass;
provided (common0).exec[4] = false and et = 0;
when ce = 0;
nextstate wait;
provided (common0).exec[4] = false and et = 0;
when ce > 0;
task pre := pre + pr;
nextstate wait;
provided (common0).exec[4] = false and et > 0;
task pre := pre + pr;
nextstate wait;

endstate;

state pass #unstable ;
set ce := 0;
task et := et + pr;
nextstate run;
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_1(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[1] and ((common0).en[3] = false);
task (common0).exec[1] := true;
nextstate schedule_airframe;

provided (common0).en[3];
task (common0).exec[3] := true;
nextstate schedule_gps;

endstate;

state schedule_airframe;
deadline eager;
provided (common0).en[3];
when (task_airframe0).ce = 0;
task (common0).exec[1] := false;
nextstate initial;
provided (common0).en[1] = false;
```

```
task (common0).exec[1] := false;
nextstate initial;

    endstate;

state schedule_gps;
provided (common0).en[3] = false;
task (common0).exec[3] := false;
nextstate initial;

    endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_2(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[5] and ((common0).en[6] = false);
task (common0).exec[5] := true;
task (common0).exec[8] := false;
task (common0).exec[9] := false;
nextstate schedule_nav_steering;

provided (common0).en[6];
task (common0).exec[6] := true;
task (common0).exec[8] := false;
task (common0).exec[9] := false;
nextstate schedule_navigator_navsteering_points;

provided (common0).en[8] and ((common0).en[5] = false) and ((common0).en[6] = false);
task (common0).exec[8] := true;
nextstate schedule_pilot_waypoints;

provided (common0).en[9] and ((common0).en[5] = false) and ((common0).en[6] = false) and
((common0).en[8] = false);
task (common0).exec[9] := true;
nextstate schedule_routes;

endstate;

state schedule_nav_steering;
deadline eager;
provided (common0).en[6];
```

```
when (task_nav_steering0).ce = 0;
task (common0).exec[5] := false;
nextstate initial;
provided (common0).en[5] = false;
task (common0).exec[5] := false;
nextstate initial;

endstate;

state schedule_navigator_navsteering_points;
provided (common0).en[6] = false;
task (common0).exec[6] := false;
nextstate initial;

endstate;

state schedule_pilot_waypoints;
/* Preemptions */
provided (common0).en[5] or (common0).en[6];
task (common0).exec[8] := false;
nextstate initial;
provided (common0).en[8] = false;
task (common0).exec[8] := false;
nextstate initial;

endstate;

state schedule_routes;
deadline eager;
provided ((common0).en[8]) and (task_routes0).et = 0;
when (task_routes0).ce = 0;
task (common0).exec[9] := false;
nextstate initial;
/* Preemptions */
provided (common0).en[5] or (common0).en[6];
task (common0).exec[9] := false;
nextstate initial;
provided (common0).en[9] = false;
task (common0).exec[9] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_3(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;
```

```
/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[0] and ((common0).en[2] = false) and ((common0).en[7] = false) and
((common0).en[10] = false);
task (common0).exec[0] := true;
nextstate schedule_af_monitor;

provided (common0).en[2] and ((common0).en[7] = false) and ((common0).en[10] = false);
task (common0).exec[2] := true;
nextstate schedule_display_device;

provided (common0).en[4] and ((common0).en[0] = false) and ((common0).en[2] = false) and
((common0).en[7] = false) and ((common0).en[10] = false);
task (common0).exec[4] := true;
nextstate schedule_nav_display;

provided (common0).en[7];
task (common0).exec[7] := true;
task (common0).exec[0] := false;
task (common0).exec[2] := false;
task (common0).exec[4] := false;
nextstate schedule_pilot_control;

provided (common0).en[10] and ((common0).en[7] = false);
task (common0).exec[10] := true;
task (common0).exec[0] := false;
task (common0).exec[2] := false;
task (common0).exec[4] := false;
nextstate schedule_tactical_steering;

endstate;

state schedule_af_monitor;
deadline eager;
provided ((common0).en[2]) and (task_af_monitor0).et = 0;
when (task_af_monitor0).ce = 0;
task (common0).exec[0] := false;
nextstate initial;
/* Preemptions */
provided (common0).en[7] or (common0).en[10];
task (common0).exec[0] := false;
nextstate initial;
provided (common0).en[0] = false;
task (common0).exec[0] := false;
nextstate initial;

endstate;

state schedule_display_device;
/* Preemptions */
provided (common0).en[7] or (common0).en[10];
task (common0).exec[2] := false;
nextstate initial;
```

```
provided (common0).en[2] = false;
task (common0).exec[2] := false;
nextstate initial;

endstate;

state schedule_nav_display;
deadline eager;
provided ((common0).en[0] or (common0).en[2]) and (task_nav_display0).et = 0;
when (task_nav_display0).ce = 0;
task (common0).exec[4] := false;
nextstate initial;
/* Preemptions */
provided (common0).en[7] or (common0).en[10];
task (common0).exec[4] := false;
nextstate initial;
provided (common0).en[4] = false;
task (common0).exec[4] := false;
nextstate initial;

endstate;

state schedule_pilot_control;
provided (common0).en[7] = false;
task (common0).exec[7] := false;
nextstate initial;

endstate;

state schedule_tactical_steering;
deadline eager;
provided (common0).en[7];
when (task_tactical_steering0).ce = 0;
task (common0).exec[10] := false;
nextstate initial;
provided (common0).en[10] = false;
task (common0).exec[10] := false;
nextstate initial;

endstate;

endprocess;

endsystem;

/*
* Observer
*
*/

pure observer safety;

state idle #start ;
deadline eager;

provided (task_af_monitor0) instate wait;
```

```
when (task_af_monitor0).cd >= dl0 - (task_af_monitor0).pre;
nextstate error;
provided (task_af_monitor0) instate run;
when (task_af_monitor0).cd >= dl0 - (task_af_monitor0).pre;
nextstate error;

provided (task_airframe0) instate wait;
when (task_airframe0).cd >= dl1;
nextstate error;
provided (task_airframe0) instate run;
when (task_airframe0).cd >= dl1;
nextstate error;

provided (task_display_device0) instate wait;
when (task_display_device0).cd >= dl2 - (task_display_device0).pre;
nextstate error;
provided (task_display_device0) instate run;
when (task_display_device0).cd >= dl2 - (task_display_device0).pre;
nextstate error;

provided (task_gps0) instate wait;
when (task_gps0).cd >= dl3;
nextstate error;
provided (task_gps0) instate run;
when (task_gps0).cd >= dl3;
nextstate error;

provided (task_nav_display0) instate wait;
when (task_nav_display0).cd >= dl4 - (task_nav_display0).pre;
nextstate error;
provided (task_nav_display0) instate run;
when (task_nav_display0).cd >= dl4 - (task_nav_display0).pre;
nextstate error;

provided (task_nav_steering0) instate wait;
when (task_nav_steering0).cd >= dl5;
nextstate error;
provided (task_nav_steering0) instate run;
when (task_nav_steering0).cd >= dl5;
nextstate error;

provided (task_navigator_navsteering_points0) instate wait;
when (task_navigator_navsteering_points0).cd >= dl6;
nextstate error;
provided (task_navigator_navsteering_points0) instate run;
when (task_navigator_navsteering_points0).cd >= dl6;
nextstate error;

provided (task_pilot_control0) instate wait;
when (task_pilot_control0).cd >= dl7;
nextstate error;
provided (task_pilot_control0) instate run;
when (task_pilot_control0).cd >= dl7;
nextstate error;

provided (task_pilot_waypoints0) instate wait;
```

```
when (task_pilot_waypoints0).cd >= dl8 - (task_pilot_waypoints0).pre;
nextstate error;
provided (task_pilot_waypoints0) instate run;
when (task_pilot_waypoints0).cd >= dl8 - (task_pilot_waypoints0).pre;
nextstate error;

provided (task_routes0) instate wait;
when (task_routes0).cd >= dl9 - (task_routes0).pre;
nextstate error;
provided (task_routes0) instate run;
when (task_routes0).cd >= dl9 - (task_routes0).pre;
nextstate error;

provided (task_tactical_steering0) instate wait;
when (task_tactical_steering0).cd >= dl10;
nextstate error;
provided (task_tactical_steering0) instate run;
when (task_tactical_steering0).cd >= dl10;
nextstate error;

endstate;

state error #error ;
endstate;

endobserver;
```

10.14 Large Distributed Non-preemptive Verimag IF Toolset Input

```
system dre;

signal event ();

/*
* Type declarations
*
*/

const pr=1;
const NTASKS=52;
type flags=array[NTASKS] of boolean;

const dl0=1000;
const dl1=1000;
const dl2=1000;
const dl3=1000;
const dl4=1000;
const dl5=1000;
const dl6=1000;
const dl7=1000;
const dl8=1000;
const dl9=1000;
const dl10=1000;
const dl11=1000;
```

```
const dl12=1000;
const dl13=1000;
const dl14=1000;
const dl15=1000;
const dl16=1000;
const dl17=1000;
const dl18=1000;
const dl19=1000;
const dl20=1000;
const dl21=1000;
const dl22=1000;
const dl23=1000;
const dl24=1000;
const dl25=1000;
const dl26=1000;
const dl27=1000;
const dl28=1000;
const dl29=1000;
const dl30=1000;
const dl31=1000;
const dl32=1000;
const dl33=1000;
const dl34=1000;
const dl35=1000;
const dl36=1000;
const dl37=1000;
const dl38=1000;
const dl39=1000;
const dl40=1000;
const dl41=1000;
const dl42=1000;
const dl43=1000;
const dl44=1000;
const dl45=1000;
const dl46=1000;
const dl47=1000;
const dl48=1000;
const dl49=1000;
const dl50=1000;
const dl51=1000;

const wcet0=2;
const wcet1=2;
const wcet2=2;
const wcet3=2;
const wcet4=2;
const wcet5=2;
const wcet6=2;
const wcet7=2;
const wcet8=2;
const wcet9=2;
const wcet10=2;
const wcet11=2;
const wcet12=2;
const wcet13=2;
const wcet14=2;
```

```
const wcet15=2;
const wcet16=2;
const wcet17=2;
const wcet18=2;
const wcet19=2;
const wcet20=2;
const wcet21=2;
const wcet22=2;
const wcet23=2;
const wcet24=2;
const wcet25=2;
const wcet26=2;
const wcet27=2;
const wcet28=2;
const wcet29=2;
const wcet30=2;
const wcet31=2;
const wcet32=2;
const wcet33=2;
const wcet34=2;
const wcet35=2;
const wcet36=2;
const wcet37=2;
const wcet38=2;
const wcet39=2;
const wcet40=2;
const wcet41=2;
const wcet42=2;
const wcet43=2;
const wcet44=2;
const wcet45=2;
const wcet46=2;
const wcet47=2;
const wcet48=2;
const wcet49=2;
const wcet50=2;
const wcet51=2;

const bcet0=1;
const bcet1=2;
const bcet2=2;
const bcet3=1;
const bcet4=2;
const bcet5=2;
const bcet6=2;
const bcet7=2;
const bcet8=1;
const bcet9=1;
const bcet10=2;
const bcet11=2;
const bcet12=2;
const bcet13=2;
const bcet14=2;
const bcet15=2;
const bcet16=2;
const bcet17=2;
```

```
const bcet18=2;
const bcet19=2;
const bcet20=2;
const bcet21=2;
const bcet22=1;
const bcet23=1;
const bcet24=1;
const bcet25=1;
const bcet26=1;
const bcet27=1;
const bcet28=2;
const bcet29=2;
const bcet30=2;
const bcet31=2;
const bcet32=2;
const bcet33=2;
const bcet34=2;
const bcet35=2;
const bcet36=2;
const bcet37=1;
const bcet38=2;
const bcet39=2;
const bcet40=2;
const bcet41=2;
const bcet42=1;
const bcet43=1;
const bcet44=1;
const bcet45=1;
const bcet46=2;
const bcet47=2;
const bcet48=2;
const bcet49=2;
const bcet50=2;
const bcet51=2;

/*
* Signalroutes
*
*
* Use signalroutes only if you have to model lossy channels
* or channels with delays.  The IF message passing builds on
* a buffered FIFO by default therefore it is suitable to
* model the real-time event channel mechanism as implemented
* in several middleware.  Try to avoid signalroutes as they
* tend to agressively increase the state space size.
*
*
* signalroute af_state_logical_1(1) #fifo #reliable #unicast #urgent;
* from task_af_state_logical_device to task_af_state_logical_display
* with event;
*
*/
signalroute airframe_1(1) #fifo #reliable #unicast #delay[1,2];
from task_airframe to task_routes
with event;
/*
```

```
*/
signalroute airframe_2(1) #fifo #reliable #unicast #delay[1,3];
from task_airframe to task_nav_steering with event;
/*
* signalroute aths_1(1) #fifo #reliable #unicast #urgent;
* from task_aths_device to task_aths_logical_device * with event;
*
* signalroute aths_2(1) #fifo #reliable #unicast #urgent;
* from task_aths_logical_device to task_aths_logical_display * with event;
*
* signalroute cnidc_1(1) #fifo #reliable #unicast #urgent;
* from task_cnidc_device to task_af_state_logical_device * with event;
*
* signalroute device_1(1) #fifo #reliable #unicast #urgent;
* from task_device_5Hz to task_tactical_physical_device_5Hz * with event;
*
* signalroute display_1(1) #fifo #reliable #unicast #urgent;
* from task_display_device to task_nav_display * with event;
*
* signalroute display_2(1) #fifo #reliable #unicast #urgent;
* from task_display_device to task_af_monitor * with event;
*
* signalroute dmt_1(1) #fifo #reliable #unicast #urgent;
* from task_dmt_device to task_dmt_logical_device * with event;
*
* signalroute dmt_2(1) #fifo #reliable #unicast #urgent;
* from task_dmt_logical_device to task_dmt_logical_display * with event;
*
* signalroute dvms_1(1) #fifo #reliable #unicast #urgent;
* from task_dvms_device to task_dvms_logical_device * with event;
*
* signalroute dvms_2(1) #fifo #reliable #unicast #urgent;
* from task_dvms_logical_device to task_dvms_logical_display * with event;
*
* signalroute ems_1(1) #fifo #reliable #unicast #urgent;
* from task_ems_device to task_ems_logical_device * with event;
*
* signalroute ems_2(1) #fifo #reliable #unicast #urgent;
* from task_ems_logical_device to task_ems_logical_display * with event;
*
* signalroute miu_1(1) #fifo #reliable #unicast #urgent;
* from task_miu_device to task_miu_logical_device * with event;
*
* signalroute miu_2(1) #fifo #reliable #unicast #urgent;
* from task_miu_logical_device to task_miu_logical_display * with event;
*
* signalroute nav_steering_1(1) #fifo #reliable #unicast #urgent;
* from task_nav_steering to task_nav_display * with event;
*
* signalroute navigator_navsteering_1(1) #fifo #reliable #unicast #urgent;
* from task_navigator_navsteering_points to task_nav_steering * with event;
*
* signalroute pilot_control_1(1) #fifo #reliable #unicast #urgent;
* from task_pilot_control to task_tactical_steering * with event;
*
* signalroute pilot_waypoints_1(1) #fifo #reliable #unicast #urgent;
```

```
* from task_pilot_waypoints to task_routes * with event;
*
* signalroute radar_1(1) #fifo #reliable #unicast #urgent;
* from task_radar_device to task_radar_logical_device * with event;
*
* signalroute radar_2(1) #fifo #reliable #unicast #urgent;
* from task_radar_logical_device to task_radar_logical_display * with event;
*
* signalroute radio1_1(1) #fifo #reliable #unicast #urgent;
* from task_radio1_device to task_radio1_logical_device * with event;
*
* signalroute radio1_2(1) #fifo #reliable #unicast #urgent;
* from task_radio1_logical_device to task_radio1_logical_display * with event;
*
* signalroute radio2_1(1) #fifo #reliable #unicast #urgent;
* from task_radio2_device to task_radio2_logical_device * with event;
*
* signalroute radio2_2(1) #fifo #reliable #unicast #urgent;
* from task_radio2_logical_device to task_radio2_logical_display * with event;
*
* signalroute rwr_1(1) #fifo #reliable #unicast #urgent;
* from task_rwr_device to task_rwr_logical_device * with event;
*
* signalroute rwr_2(1) #fifo #reliable #unicast #urgent;
* from task_rwr_logical_device to task_rwr_logical_display * with event;
*
* signalroute saahs_1(1) #fifo #reliable #unicast #urgent;
* from task_saahs_device to task_af_state_logical_device * with event;
*
* signalroute tactical_logical_1(1) #fifo #reliable #unicast #urgent;
* from task_tactical_logical_device to task_tactical_mode * with event;
*
* signalroute tactical_logical_2(1) #fifo #reliable #unicast #urgent;
* from task_tactical_logical_device to task_airframe * with event;
*
* signalroute tactical_mode_1(1) #fifo #reliable #unicast #urgent;
* from task_tactical_mode to task_phase_manager * with event;
*
* signalroute tactical_physical_1(1) #fifo #reliable #unicast #urgent;
* from task_tactical_physical_device_5Hz to task_tactical_logical_device * with event;
*
* signalroute tactical_steering_1(1) #fifo #reliable #unicast #urgent;
* from task_tactical_steering to task_nav_display * with event;
*
* signalroute tacts_1(1) #fifo #reliable #unicast #urgent;
* from task_tacts_device to task_tacts_logical_device * with event;
*
* signalroute tacts_2(1) #fifo #reliable #unicast #urgent;
* from task_tacts_logical_device to task_tacts_logical_display * with event;
*
* signalroute wmc_1(1) #fifo #reliable #unicast #urgent;
* from task_wmc_device to task_wmc_logical_device * with event;
*
* signalroute wmc_2(1) #fifo #reliable #unicast #urgent;
* from task_wmc_logical_device to task_wmc_logical_display * with event;
*
```

```
*/

/*
* Common variables
*
* We need a separate process to store them...
*/

process common(1);

var en flags public;
var exec flags public;

endprocess;

/*
* Timer - RateGen_10Hz
*
*/

process timer_RateGen_10Hz(1);

var ce clock;
const period_RateGen_10Hz=100;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_10Hz;
output event() to task_display_device0;
output event() to task_miu_device0;
output event() to task_pilot_control0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 0 - af_monitor
*
*/

process task_af_monitor(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
```

```
set cd := 0;
task (common0).en[0] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[0];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet0 and ce >= bcet0;
task (common0).en[0] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[0] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 9 - display_device
*
*/

process task_display_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[9] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[9];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet9 and ce >= bcet9;
task (common0).en[9] := false;
```

```
reset ce;
reset cd;
output event() /* via display_10 */ to task_nav_display0;
output event() /* via display_20 */ to task_af_monitor0;
nextstate initial;
deadline eager;
provided (common0).exec[9] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 19 - miu_device
*
*/

process task_miu_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[19] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[19];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet19 and ce >= bcet19;
task (common0).en[19] := false;
reset ce;
reset cd;
output event() /* via miu_10 */ to task_miu_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[19] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 20 - miu_logical_device
```

```
*
*/

process task_miu_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[20] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[20];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet20 and ce >= bcet20;
task (common0).en[20] := false;
reset ce;
reset cd;
output event() /* via miu_20 */ to task_miu_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[20] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 21 - miu_logical_display
*
*/

process task_miu_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[21] := true;
nextstate wait;
endstate;
```

```
state wait;
deadline eager;
provided (common0).exec[21];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet21 and ce >= bcet21;
task (common0).en[21] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[21] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 22 - nav_display
*
*/

process task_nav_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[22] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[22];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet22 and ce >= bcet22;
task (common0).en[22] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
```

```
provided (common0).exec[22] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 26 - pilot_control
*
*/

process task_pilot_control(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[26] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[26];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet26 and ce >= bcet26;
task (common0).en[26] := false;
reset ce;
reset cd;
output event() /* via pilot_control_10 */ to task_tactical_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[26] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 45 - tactical_steering
*
*/

process task_tactical_steering(1);

var ce clock public;
```

```
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[45] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[45];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet45 and ce >= bcet45;
task (common0).en[45] := false;
reset ce;
reset cd;
output event() /* via tactical_steering_10 */ to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[45] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_1Hz
*
*/

process timer_RateGen_1Hz(1);

var ce clock;
const period_RateGen_1Hz=1000;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_1Hz;
output event() to task_aths_device0;
output event() to task_cnidc_device0;
output event() to task_dvms_device0;
output event() to task_saahs_device0;
```

```
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 1 - af_state_logical_device
*
*/

process task_af_state_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[1] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[1];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet1 and ce >= bcet1;
task (common0).en[1] := false;
reset ce;
reset cd;
output event() /* via af_state_logical_10 */ to task_af_state_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[1] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 2 - af_state_logical_display
*
*/

process task_af_state_logical_display(1);

var ce clock public;
var cd clock public;
```

```
state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[2] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[2];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet2 and ce >= bcet2;
task (common0).en[2] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[2] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 4 - aths_device
*
*/

process task_aths_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[4] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[4];
set ce := 0;
nextstate run;
endstate;
```

```
state run;
deadline delayable;
when ce <= wcet4 and ce >= bcet4;
task (common0).en[4] := false;
reset ce;
reset cd;
output event() /* via aths_10 */ to task_aths_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[4] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 5 - aths_logical_device
*
*/

process task_aths_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[5] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[5];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet5 and ce >= bcet5;
task (common0).en[5] := false;
reset ce;
reset cd;
output event() /* via aths_20 */ to task_aths_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[5] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Task 6 - aths_logical_display
*
*/

process task_aths_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[6] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[6];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet6 and ce >= bcet6;
task (common0).en[6] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[6] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 7 - cnidc_device
*
*/

process task_cnidc_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[7] := true;
```

```
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[7];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet7 and ce >= bcet7;
task (common0).en[7] := false;
reset ce;
reset cd;
output event() /* via cnidc_10 */ to task_af_state_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[7] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 13 - dvms_device
*
*/

process task_dvms_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[13] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[13];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet13 and ce >= bcet13;
task (common0).en[13] := false;
reset ce;
```

```
reset cd;
output event() /* via dvms_10 */ to task_dvms_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[13] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 14 - dvms_logical_device
*
*/

process task_dvms_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[14] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[14];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet14 and ce >= bcet14;
task (common0).en[14] := false;
reset ce;
reset cd;
output event() /* via dvms_20 */ to task_dvms_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[14] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 15 - dvms_logical_display
*
*/
```

```
process task_dvms_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[15] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[15];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet15 and ce >= bcet15;
task (common0).en[15] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[15] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 41 - saahs_device
*
*/

process task_saahs_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[41] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
```

```
provided (common0).exec[41];
set ce := 0;
nextstate run;
endstate;

    state run;
deadline delayable;
when ce <= wcet41 and ce >= bcet41;
task (common0).en[41] := false;
reset ce;
reset cd;
output event() /* via saahs_10 */ to task_af_state_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[41] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_20Hz
*
*/

process timer_RateGen_20Hz(1);

var ce clock;
const period_RateGen_20Hz=50;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_20Hz;
output event() to task_ems_device0;
output event() to task_radar_device0;
output event() to task_rwr_device0;
output event() to task_tacts_device0;
output event() to task_wmc_device0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 16 - ems_device
*
*/
```

```
process task_ems_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[16] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[16];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet16 and ce >= bcet16;
task (common0).en[16] := false;
reset ce;
reset cd;
output event() /* via ems_10 */ to task_ems_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[16] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 17 - ems_logical_device
*
*/

process task_ems_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[17] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
```

```
provided (common0).exec[17];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet17 and ce >= bcet17;
task (common0).en[17] := false;
reset ce;
reset cd;
output event() /* via ems_20 */ to task_ems_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[17] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 18 - ems_logical_display
*
*/

process task_ems_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[18] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[18];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet18 and ce >= bcet18;
task (common0).en[18] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[18] = false;
when ce = 0;
```

```
nextstate wait;
endstate;

endprocess;

/*
* Task 28 - radar_device
*
*/

process task_radar_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[28] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[28];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet28 and ce >= bcet28;
task (common0).en[28] := false;
reset ce;
reset cd;
output event() /* via radar_10 */ to task_radar_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[28] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 29 - radar_logical_device
*
*/

process task_radar_logical_device(1);

var ce clock public;
var cd clock public;
```

```
state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[29] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[29];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet29 and ce >= bcet29;
task (common0).en[29] := false;
reset ce;
reset cd;
output event() /* via radar_20 */ to task_radar_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[29] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 30 - radar_logical_display
*
*/

process task_radar_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[30] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[30];
set ce := 0;
nextstate run;
endstate;
```

```
state run;
deadline delayable;
when ce <= wcet30 and ce >= bcet30;
task (common0).en[30] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[30] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 38 - rwr_device
*
*/

process task_rwr_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[38] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[38];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet38 and ce >= bcet38;
task (common0).en[38] := false;
reset ce;
reset cd;
output event() /* via rwr_10 */ to task_rwr_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[38] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Task 39 - rwr_logical_device
*
*/

process task_rwr_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[39] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[39];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet39 and ce >= bcet39;
task (common0).en[39] := false;
reset ce;
reset cd;
output event() /* via rwr_20 */ to task_rwr_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[39] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 40 - rwr_logical_display
*
*/

process task_rwr_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[40] := true;
```

```
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[40];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet40 and ce >= bcet40;
task (common0).en[40] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[40] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 46 - tacts_device
*
*/

process task_tacts_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[46] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[46];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet46 and ce >= bcet46;
task (common0).en[46] := false;
reset ce;
reset cd;
```

```
output event() /* via tacts_10 */ to task_tacts_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[46] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 47 - tacts_logical_device
*
*/

process task_tacts_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[47] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[47];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet47 and ce >= bcet47;
task (common0).en[47] := false;
reset ce;
reset cd;
output event() /* via tacts_20 */ to task_tacts_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[47] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 48 - tacts_logical_display
*
*/
```

```
process task_tacts_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[48] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[48];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet48 and ce >= bcet48;
task (common0).en[48] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[48] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 49 - wmc_device
*
*/

process task_wmc_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[49] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[49];
```

```
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet49 and ce >= bcet49;
task (common0).en[49] := false;
reset ce;
reset cd;
output event() /* via wmc_10 */ to task_wmc_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[49] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 50 - wmc_logical_device
*
*/

process task_wmc_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[50] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[50];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet50 and ce >= bcet50;
task (common0).en[50] := false;
reset ce;
reset cd;
output event() /* via wmc_20 */ to task_wmc_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[50] = false;
when ce = 0;
```

```
nextstate wait;
endstate;

endprocess;

/*
* Task 51 - wmc_logical_display
*
*/

process task_wmc_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[51] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[51];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet51 and ce >= bcet51;
task (common0).en[51] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[51] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_40Hz
*
*/

process timer_RateGen_40Hz(1);

var ce clock;
const period_RateGen_40Hz=25;

state start #start ;
```

```
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_40Hz;
output event() to task_dmt_device0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
* Task 10 - dmt_device
*
*/

process task_dmt_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[10] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[10];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet10 and ce >= bcet10;
task (common0).en[10] := false;
reset ce;
reset cd;
output event() /* via dmt_10 */ to task_dmt_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[10] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Task 11 - dmt_logical_device
*
*/

process task_dmt_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[11] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[11];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet11 and ce >= bcet11;
task (common0).en[11] := false;
reset ce;
reset cd;
output event() /* via dmt_20 */ to task_dmt_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[11] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 12 - dmt_logical_display
*
*/

process task_dmt_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[12] := true;
```

```
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[12];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet12 and ce >= bcet12;
task (common0).en[12] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[12] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Timer - RateGen_5Hz
*
*/

process timer_RateGen_5Hz(1);

var ce clock;
const period_RateGen_5Hz=200;

state start #start ;
deadline eager;
set ce := 0;
nextstate timer;
endstate;

state timer;
deadline eager;
when ce = period_RateGen_5Hz;
output event() to task_device_5Hz0;
output event() to task_navigator_navsteering_points0;
output event() to task_pilot_waypoints0;
output event() to task_radio1_device0;
output event() to task_radio2_device0;
set ce := 0;
nextstate timer;
endstate;

endprocess;

/*
```

```
* Task 3 - airframe
*
*/

process task_airframe(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[3] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[3];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet3 and ce >= bcet3;
task (common0).en[3] := false;
reset ce;
reset cd;
output event() via airframe_10 to task_routes0;
output event() via airframe_20 to task_nav_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[3] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 8 - device_5Hz
*
*/

process task_device_5Hz(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[8] := true;
```

```
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[8];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet8 and ce >= bcet8;
task (common0).en[8] := false;
reset ce;
reset cd;
output event() /* via device_10 */ to task_tactical_physical_device_5Hz0;
nextstate initial;
deadline eager;
provided (common0).exec[8] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 23 - nav_steering
*
*/

process task_nav_steering(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[23] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[23];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet23 and ce >= bcet23;
task (common0).en[23] := false;
reset ce;
```

```
reset cd;
output event() /* via nav_steering_10 */ to task_nav_display0;
nextstate initial;
deadline eager;
provided (common0).exec[23] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 24 - navigator_navsteering_points
*
*/

process task_navigator_navsteering_points(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[24] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[24];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet24 and ce >= bcet24;
task (common0).en[24] := false;
reset ce;
reset cd;
output event() /* via navigator_navsteering_10 */ to task_nav_steering0;
nextstate initial;
deadline eager;
provided (common0).exec[24] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 25 - phase_manager
*
*/
```

```
process task_phase_manager(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[25] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[25];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet25 and ce >= bcet25;
task (common0).en[25] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[25] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 27 - pilot_waypoints
*
*/

process task_pilot_waypoints(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[27] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
```

```
provided (common0).exec[27];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet27 and ce >= bcet27;
task (common0).en[27] := false;
reset ce;
reset cd;
output event() /* via pilot_waypoints_10 */ to task_routes0;
nextstate initial;
deadline eager;
provided (common0).exec[27] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 31 - radio1_device
*
*/

process task_radio1_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[31] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[31];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet31 and ce >= bcet31;
task (common0).en[31] := false;
reset ce;
reset cd;
output event() /* via radio1_10 */ to task_radio1_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[31] = false;
```

```
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 32 - radio1_logical_device
*
*/

process task_radio1_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[32] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[32];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet32 and ce >= bcet32;
task (common0).en[32] := false;
reset ce;
reset cd;
output event() /* via radio1_20 */ to task_radio1_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[32] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 33 - radio1_logical_display
*
*/

process task_radio1_logical_display(1);

var ce clock public;
var cd clock public;
```

```
state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[33] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[33];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet33 and ce >= bcet33;
task (common0).en[33] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[33] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 34 - radio2_device
*
*/

process task_radio2_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[34] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[34];
set ce := 0;
nextstate run;
endstate;
```

```
state run;
deadline delayable;
when ce <= wcet34 and ce >= bcet34;
task (common0).en[34] := false;
reset ce;
reset cd;
output event() /* via radio2_10 */ to task_radio2_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[34] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 35 - radio2_logical_device
*
*/

process task_radio2_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[35] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[35];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet35 and ce >= bcet35;
task (common0).en[35] := false;
reset ce;
reset cd;
output event() /* via radio2_20 */ to task_radio2_logical_display0;
nextstate initial;
deadline eager;
provided (common0).exec[35] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;
```

```
/*
* Task 36 - radio2_logical_display
*
*/

process task_radio2_logical_display(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[36] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[36];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet36 and ce >= bcet36;
task (common0).en[36] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[36] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 37 - routes
*
*/

process task_routes(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[37] := true;
```

```
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[37];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet37 and ce >= bcet37;
task (common0).en[37] := false;
reset ce;
reset cd;
nextstate initial;
deadline eager;
provided (common0).exec[37] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 42 - tactical_logical_device
*
*/

process task_tactical_logical_device(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[42] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[42];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet42 and ce >= bcet42;
task (common0).en[42] := false;
reset ce;
reset cd;
```

```
output event() /* via tactical_logical_10 */ to task_tactical_mode0;
output event() /* via tactical_logical_20 */ to task_airframe0;
nextstate initial;
deadline eager;
provided (common0).exec[42] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 43 - tactical_mode
*
*/

process task_tactical_mode(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[43] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[43];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet43 and ce >= bcet43;
task (common0).en[43] := false;
reset ce;
reset cd;
output event() /* via tactical_mode_10 */ to task_phase_manager0;
nextstate initial;
deadline eager;
provided (common0).exec[43] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Task 44 - tactical_physical_device_5Hz
*
*/
```

```
process task_tactical_physical_device_5Hz(1);

var ce clock public;
var cd clock public;

state initial #start ;
deadline eager;
input event ();
set cd := 0;
task (common0).en[44] := true;
nextstate wait;
endstate;

state wait;
deadline eager;
provided (common0).exec[44];
set ce := 0;
nextstate run;
endstate;

state run;
deadline delayable;
when ce <= wcet44 and ce >= bcet44;
task (common0).en[44] := false;
reset ce;
reset cd;
output event() /* via tactical_physical_10 */ to task_tactical_logical_device0;
nextstate initial;
deadline eager;
provided (common0).exec[44] = false;
when ce = 0;
nextstate wait;
endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_10Hz(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[0] and ((common0).en[9] = false) and ((common0).en[19] = false)
and ((common0).en[20] = false) and ((common0).en[21] = false) and ((common0).en[22] =
false);
```

```
task (common0).exec[0] := true;
nextstate schedule_af_monitor;

provided (common0).en[9] and ((common0).en[19] = false) and ((common0).en[20] = false)
and ((common0).en[21] = false);
task (common0).exec[9] := true;
nextstate schedule_display_device;

provided (common0).en[19];
task (common0).exec[19] := true;
nextstate schedule_miu_device;

provided (common0).en[20] and ((common0).en[19] = false);
task (common0).exec[20] := true;
nextstate schedule_miu_logical_device;

provided (common0).en[21] and ((common0).en[19] = false) and ((common0).en[20] = false);
task (common0).exec[21] := true;
nextstate schedule_miu_logical_display;

provided (common0).en[22] and ((common0).en[9] = false) and ((common0).en[19] = false)
and ((common0).en[20] = false) and ((common0).en[21] = false);
task (common0).exec[22] := true;
nextstate schedule_nav_display;

provided (common0).en[26] and ((common0).en[0] = false) and ((common0).en[9] = false)
and ((common0).en[19] = false) and ((common0).en[20] = false) and ((common0).en[21] =
false) and ((common0).en[22] = false);
task (common0).exec[26] := true;
nextstate schedule_pilot_control;

provided (common0).en[45] and ((common0).en[0] = false) and ((common0).en[9] = false)
and ((common0).en[19] = false) and ((common0).en[20] = false) and ((common0).en[21] =
false) and ((common0).en[22] = false) and ((common0).en[26] = false);
task (common0).exec[45] := true;
nextstate schedule_tactical_steering;

endstate;

state schedule_af_monitor;
deadline eager;
provided (common0).en[9] or (common0).en[19] or (common0).en[20] or (common0).en[21] or
(common0).en[22];
when (task_af_monitor0).ce = 0;
task (common0).exec[0] := false;
nextstate initial;
provided (common0).en[0] = false;
task (common0).exec[0] := false;
nextstate initial;

endstate;

state schedule_display_device;
deadline eager;
provided (common0).en[19] or (common0).en[20] or (common0).en[21];
when (task_display_device0).ce = 0;
```

```
task (common0).exec[9] := false;
nextstate initial;
provided (common0).en[9] = false;
task (common0).exec[9] := false;
nextstate initial;

endstate;

state schedule_miu_device;
provided (common0).en[19] = false;
task (common0).exec[19] := false;
nextstate initial;

endstate;

state schedule_miu_logical_device;
deadline eager;
provided (common0).en[19];
when (task_miu_logical_device0).ce = 0;
task (common0).exec[20] := false;
nextstate initial;
provided (common0).en[20] = false;
task (common0).exec[20] := false;
nextstate initial;

endstate;

state schedule_miu_logical_display;
deadline eager;
provided (common0).en[19] or (common0).en[20];
when (task_miu_logical_display0).ce = 0;
task (common0).exec[21] := false;
nextstate initial;
provided (common0).en[21] = false;
task (common0).exec[21] := false;
nextstate initial;

endstate;

state schedule_nav_display;
deadline eager;
provided (common0).en[9] or (common0).en[19] or (common0).en[20] or (common0).en[21];
when (task_nav_display0).ce = 0;
task (common0).exec[22] := false;
nextstate initial;
provided (common0).en[22] = false;
task (common0).exec[22] := false;
nextstate initial;

endstate;

state schedule_pilot_control;
deadline eager;
provided (common0).en[0] or (common0).en[9] or (common0).en[19] or (common0).en[20] or
(common0).en[21] or (common0).en[22];
when (task_pilot_control0).ce = 0;
```

```
task (common0).exec[26] := false;
nextstate initial;
provided (common0).en[26] = false;
task (common0).exec[26] := false;
nextstate initial;

endstate;

state schedule_tactical_steering;
deadline eager;
provided (common0).en[0] or (common0).en[9] or (common0).en[19] or (common0).en[20] or
(common0).en[21] or (common0).en[22] or (common0).en[26];
when (task_tactical_steering0).ce = 0;
task (common0).exec[45] := false;
nextstate initial;
provided (common0).en[45] = false;
task (common0).exec[45] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_1Hz(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[1] and ((common0).en[4] = false) and ((common0).en[5] = false) and
((common0).en[6] = false) and ((common0).en[7] = false) and ((common0).en[13] = false)
and ((common0).en[14] = false) and ((common0).en[15] = false) and ((common0).en[41] =
false);
task (common0).exec[1] := true;
nextstate schedule_af_state_logical_device;

provided (common0).en[2] and ((common0).en[1] = false) and ((common0).en[4] = false) and
((common0).en[5] = false) and ((common0).en[6] = false) and ((common0).en[7] = false)
and ((common0).en[13] = false) and ((common0).en[14] = false) and ((common0).en[15] =
false) and ((common0).en[41] = false);
task (common0).exec[2] := true;
nextstate schedule_af_state_logical_display;

provided (common0).en[4];
task (common0).exec[4] := true;
nextstate schedule_aths_device;
```

```
provided (common0).en[5] and ((common0).en[4] = false);
task (common0).exec[5] := true;
nextstate schedule_aths_logical_device;

provided (common0).en[6] and ((common0).en[4] = false) and ((common0).en[5] = false);
task (common0).exec[6] := true;
nextstate schedule_aths_logical_display;

provided (common0).en[7] and ((common0).en[4] = false) and ((common0).en[5] = false) and
((common0).en[6] = false) and ((common0).en[13] = false) and ((common0).en[14] = false)
and ((common0).en[15] = false) and ((common0).en[41] = false);
task (common0).exec[7] := true;
nextstate schedule_cnidc_device;

provided (common0).en[13] and ((common0).en[4] = false) and ((common0).en[5] = false)
and ((common0).en[6] = false);
task (common0).exec[13] := true;
nextstate schedule_dvms_device;

provided (common0).en[14] and ((common0).en[4] = false) and ((common0).en[5] = false)
and ((common0).en[6] = false) and ((common0).en[13] = false);
task (common0).exec[14] := true;
nextstate schedule_dvms_logical_device;

provided (common0).en[15] and ((common0).en[4] = false) and ((common0).en[5] = false)
and ((common0).en[6] = false) and ((common0).en[13] = false) and ((common0).en[14] = false);
task (common0).exec[15] := true;
nextstate schedule_dvms_logical_display;

provided (common0).en[41] and ((common0).en[4] = false) and ((common0).en[5] = false)
and ((common0).en[6] = false) and ((common0).en[13] = false) and ((common0).en[14] = false)
and ((common0).en[15] = false);
task (common0).exec[41] := true;
nextstate schedule_saahs_device;

endstate;

state schedule_af_state_logical_device;
deadline eager;
provided (common0).en[4] or (common0).en[5] or (common0).en[6] or (common0).en[7] or (common0).en[13]
or (common0).en[14] or (common0).en[15] or (common0).en[41];
when (task_af_state_logical_device0).ce = 0;
task (common0).exec[1] := false;
nextstate initial;
provided (common0).en[1] = false;
task (common0).exec[1] := false;
nextstate initial;

endstate;

state schedule_af_state_logical_display;
deadline eager;
provided (common0).en[1] or (common0).en[4] or (common0).en[5] or (common0).en[6] or (common0).en[7]
or (common0).en[13] or (common0).en[14] or (common0).en[15] or (common0).en[41];
when (task_af_state_logical_display0).ce = 0;
```

```
task (common0).exec[2] := false;
nextstate initial;
provided (common0).en[2] = false;
task (common0).exec[2] := false;
nextstate initial;

endstate;

state schedule_aths_device;
provided (common0).en[4] = false;
task (common0).exec[4] := false;
nextstate initial;

endstate;

state schedule_aths_logical_device;
deadline eager;
provided (common0).en[4];
when (task_aths_logical_device0).ce = 0;
task (common0).exec[5] := false;
nextstate initial;
provided (common0).en[5] = false;
task (common0).exec[5] := false;
nextstate initial;

endstate;

state schedule_aths_logical_display;
deadline eager;
provided (common0).en[4] or (common0).en[5];
when (task_aths_logical_display0).ce = 0;
task (common0).exec[6] := false;
nextstate initial;
provided (common0).en[6] = false;
task (common0).exec[6] := false;
nextstate initial;

endstate;

state schedule_cnidc_device;
deadline eager;
provided (common0).en[4] or (common0).en[5] or (common0).en[6] or (common0).en[13] or
(common0).en[14] or (common0).en[15] or (common0).en[41];
when (task_cnidc_device0).ce = 0;
task (common0).exec[7] := false;
nextstate initial;
provided (common0).en[7] = false;
task (common0).exec[7] := false;
nextstate initial;

endstate;

state schedule_dvms_device;
deadline eager;
provided (common0).en[4] or (common0).en[5] or (common0).en[6];
when (task_dvms_device0).ce = 0;
```

```
task (common0).exec[13] := false;
nextstate initial;
provided (common0).en[13] = false;
task (common0).exec[13] := false;
nextstate initial;

endstate;

state schedule_dvms_logical_device;
deadline eager;
provided (common0).en[4] or (common0).en[5] or (common0).en[6] or (common0).en[13];
when (task_dvms_logical_device0).ce = 0;
task (common0).exec[14] := false;
nextstate initial;
provided (common0).en[14] = false;
task (common0).exec[14] := false;
nextstate initial;

endstate;

state schedule_dvms_logical_display;
deadline eager;
/*
* We have to hack the scheduler to compensate for
* the race conditions caused by non-atomic broadcasts...
*
*/
provided (common0).en[4] or (common0).en[5] or (common0).en[6] or (common0).en[13] or
(common0).en[14];
when (task_dvms_logical_display0).ce = 0;
task (common0).exec[15] := false;
nextstate initial;
provided (common0).en[15] = false;
task (common0).exec[15] := false;
nextstate initial;

endstate;

state schedule_saahs_device;
deadline eager;
/*
* We have to hack the scheduler to compensate for
* the race conditions caused by non-atomic broadcasts...
*
*/
provided (common0).en[4] or (common0).en[5] or (common0).en[6] or (common0).en[13] or
(common0).en[14] or (common0).en[15];
when (task_saahs_device0).ce = 0;
task (common0).exec[41] := false;
nextstate initial;
provided (common0).en[41] = false;
task (common0).exec[41] := false;
nextstate initial;

endstate;
```

```
endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_20Hz(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[16];
task (common0).exec[16] := true;
nextstate schedule_ems_device;

provided (common0).en[17] and ((common0).en[16] = false);
task (common0).exec[17] := true;
nextstate schedule_ems_logical_device;

provided (common0).en[18] and ((common0).en[16] = false) and ((common0).en[17] = false);
task (common0).exec[18] := true;
nextstate schedule_ems_logical_display;

provided (common0).en[28] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false);
task (common0).exec[28] := true;
nextstate schedule_radar_device;

provided (common0).en[29] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false);
task (common0).exec[29] := true;
nextstate schedule_radar_logical_device;

provided (common0).en[30] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false);
task (common0).exec[30] := true;
nextstate schedule_radar_logical_display;

provided (common0).en[38] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false);
task (common0).exec[38] := true;
nextstate schedule_rwr_device;

provided (common0).en[39] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false);
task (common0).exec[39] := true;
```

```
nextstate schedule_rwr_logical_device;

provided (common0).en[40] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false) and ((common0).en[39]
= false);
task (common0).exec[40] := true;
nextstate schedule_rwr_logical_display;

provided (common0).en[46] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false) and ((common0).en[39]
= false) and ((common0).en[40] = false);
task (common0).exec[46] := true;
nextstate schedule_tacts_device;

provided (common0).en[47] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false) and ((common0).en[39]
= false) and ((common0).en[40] = false) and ((common0).en[46] = false);
task (common0).exec[47] := true;
nextstate schedule_tacts_logical_device;

provided (common0).en[48] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false) and ((common0).en[39]
= false) and ((common0).en[40] = false) and ((common0).en[46] = false) and ((common0).en[47]
= false);
task (common0).exec[48] := true;
nextstate schedule_tacts_logical_display;

provided (common0).en[49] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false) and ((common0).en[39]
= false) and ((common0).en[40] = false) and ((common0).en[46] = false) and ((common0).en[47]
= false) and ((common0).en[48] = false);
task (common0).exec[49] := true;
nextstate schedule_wmc_device;

provided (common0).en[50] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false) and ((common0).en[39]
= false) and ((common0).en[40] = false) and ((common0).en[46] = false) and ((common0).en[47]
= false) and ((common0).en[48] = false) and ((common0).en[49] = false);
task (common0).exec[50] := true;
nextstate schedule_wmc_logical_device;

provided (common0).en[51] and ((common0).en[16] = false) and ((common0).en[17] = false)
and ((common0).en[18] = false) and ((common0).en[28] = false) and ((common0).en[29] =
false) and ((common0).en[30] = false) and ((common0).en[38] = false) and ((common0).en[39]
= false) and ((common0).en[40] = false) and ((common0).en[46] = false) and ((common0).en[47]
= false) and ((common0).en[48] = false) and ((common0).en[49] = false) and ((common0).en[50]
= false);
task (common0).exec[51] := true;
nextstate schedule_wmc_logical_display;
```

```
endstate;

state schedule_ems_device;
provided (common0).en[16] = false;
task (common0).exec[16] := false;
nextstate initial;

endstate;

state schedule_ems_logical_device;
deadline eager;
provided (common0).en[16];
when (task_ems_logical_device0).ce = 0;
task (common0).exec[17] := false;
nextstate initial;
provided (common0).en[17] = false;
task (common0).exec[17] := false;
nextstate initial;

endstate;

state schedule_ems_logical_display;
deadline eager;
provided (common0).en[16] or (common0).en[17];
when (task_ems_logical_display0).ce = 0;
task (common0).exec[18] := false;
nextstate initial;
provided (common0).en[18] = false;
task (common0).exec[18] := false;
nextstate initial;

endstate;

state schedule_radar_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18];
when (task_radar_device0).ce = 0;
task (common0).exec[28] := false;
nextstate initial;
provided (common0).en[28] = false;
task (common0).exec[28] := false;
nextstate initial;

endstate;

state schedule_radar_logical_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28];
when (task_radar_logical_device0).ce = 0;
task (common0).exec[29] := false;
nextstate initial;
provided (common0).en[29] = false;
task (common0).exec[29] := false;
nextstate initial;

endstate;
```

```
state schedule_radar_logical_display;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29];
when (task_radar_logical_display0).ce = 0;
task (common0).exec[30] := false;
nextstate initial;
provided (common0).en[30] = false;
task (common0).exec[30] := false;
nextstate initial;

endstate;

state schedule_rwr_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30];
when (task_rwr_device0).ce = 0;
task (common0).exec[38] := false;
nextstate initial;
provided (common0).en[38] = false;
task (common0).exec[38] := false;
nextstate initial;

endstate;

state schedule_rwr_logical_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30] or (common0).en[38];
when (task_rwr_logical_device0).ce = 0;
task (common0).exec[39] := false;
nextstate initial;
provided (common0).en[39] = false;
task (common0).exec[39] := false;
nextstate initial;

endstate;

state schedule_rwr_logical_display;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30] or (common0).en[38] or (common0).en[39];
when (task_rwr_logical_display0).ce = 0;
task (common0).exec[40] := false;
nextstate initial;
provided (common0).en[40] = false;
task (common0).exec[40] := false;
nextstate initial;

endstate;

state schedule_tacts_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
```

```
or (common0).en[29] or (common0).en[30] or (common0).en[38] or (common0).en[39] or (common0).en[40];
when (task_tacts_device0).ce = 0;
task (common0).exec[46] := false;
nextstate initial;
provided (common0).en[46] = false;
task (common0).exec[46] := false;
nextstate initial;

endstate;

state schedule_tacts_logical_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30] or (common0).en[38] or (common0).en[39] or (common0).en[40]
or (common0).en[46];
when (task_tacts_logical_device0).ce = 0;
task (common0).exec[47] := false;
nextstate initial;
provided (common0).en[47] = false;
task (common0).exec[47] := false;
nextstate initial;

endstate;

state schedule_tacts_logical_display;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30] or (common0).en[38] or (common0).en[39] or (common0).en[40]
or (common0).en[46] or (common0).en[47];
when (task_tacts_logical_display0).ce = 0;
task (common0).exec[48] := false;
nextstate initial;
provided (common0).en[48] = false;
task (common0).exec[48] := false;
nextstate initial;

endstate;

state schedule_wmc_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30] or (common0).en[38] or (common0).en[39] or (common0).en[40]
or (common0).en[46] or (common0).en[47] or (common0).en[48];
when (task_wmc_device0).ce = 0;
task (common0).exec[49] := false;
nextstate initial;
provided (common0).en[49] = false;
task (common0).exec[49] := false;
nextstate initial;

endstate;

state schedule_wmc_logical_device;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30] or (common0).en[38] or (common0).en[39] or (common0).en[40]
```

```
or (common0).en[46] or (common0).en[47] or (common0).en[48] or (common0).en[49];
when (task_wmc_logical_device0).ce = 0;
task (common0).exec[50] := false;
nextstate initial;
provided (common0).en[50] = false;
task (common0).exec[50] := false;
nextstate initial;

endstate;

state schedule_wmc_logical_display;
deadline eager;
provided (common0).en[16] or (common0).en[17] or (common0).en[18] or (common0).en[28]
or (common0).en[29] or (common0).en[30] or (common0).en[38] or (common0).en[39] or (common0).en[40]
or (common0).en[46] or (common0).en[47] or (common0).en[48] or (common0).en[49] or (common0).en[50];
when (task_wmc_logical_display0).ce = 0;
task (common0).exec[51] := false;
nextstate initial;
provided (common0).en[51] = false;
task (common0).exec[51] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_40Hz(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[10];
task (common0).exec[10] := true;
nextstate schedule_dmt_device;

    provided (common0).en[11] and ((common0).en[10] = false);
task (common0).exec[11] := true;
nextstate schedule_dmt_logical_device;

    provided (common0).en[12] and ((common0).en[10] = false) and ((common0).en[11] = false);
task (common0).exec[12] := true;
nextstate schedule_dmt_logical_display;

endstate;
```

```
state schedule_dmt_device;
provided (common0).en[10] = false;
task (common0).exec[10] := false;
nextstate initial;

endstate;

state schedule_dmt_logical_device;
deadline eager;
provided (common0).en[10];
when (task_dmt_logical_device0).ce = 0;
task (common0).exec[11] := false;
nextstate initial;
provided (common0).en[11] = false;
task (common0).exec[11] := false;
nextstate initial;

endstate;

state schedule_dmt_logical_display;
deadline eager;
provided (common0).en[10] or (common0).en[11];
when (task_dmt_logical_display0).ce = 0;
task (common0).exec[12] := false;
nextstate initial;
provided (common0).en[12] = false;
task (common0).exec[12] := false;
nextstate initial;

endstate;

endprocess;

/*
* Fixed-priority scheduler
*
*/

process scheduler_CPU_5Hz(1);

state start #start ;
deadline eager;
nextstate initial;
endstate;

/* Scheduling enabled tasks */

state initial;
deadline eager;
provided (common0).en[3] and ((common0).en[8] = false) and ((common0).en[25] = false)
and ((common0).en[31] = false) and ((common0).en[32] = false) and ((common0).en[33] =
false) and ((common0).en[34] = false) and ((common0).en[35] = false) and ((common0).en[36]
= false) and ((common0).en[42] = false) and ((common0).en[43] = false) and ((common0).en[44]
= false);
task (common0).exec[3] := true;
nextstate schedule_airframe;
```

```
provided (common0).en[8] and ((common0).en[31] = false) and ((common0).en[32] = false)
and ((common0).en[33] = false) and ((common0).en[34] = false) and ((common0).en[35] =
false) and ((common0).en[36] = false);
task (common0).exec[8] := true;
nextstate schedule_device_5Hz;

provided (common0).en[23] and ((common0).en[3] = false) and ((common0).en[8] = false)
and ((common0).en[24] = false) and ((common0).en[25] = false) and ((common0).en[27] =
false) and ((common0).en[31] = false) and ((common0).en[32] = false) and ((common0).en[33]
= false) and ((common0).en[34] = false) and ((common0).en[35] = false) and ((common0).en[36]
= false) and ((common0).en[37] = false) and ((common0).en[42] = false) and ((common0).en[43]
= false) and ((common0).en[44] = false);
task (common0).exec[23] := true;
nextstate schedule_nav_steering;

provided (common0).en[24] and ((common0).en[3] = false) and ((common0).en[8] = false)
and ((common0).en[25] = false) and ((common0).en[27] = false) and ((common0).en[31] =
false) and ((common0).en[32] = false) and ((common0).en[33] = false) and ((common0).en[34]
= false) and ((common0).en[35] = false) and ((common0).en[36] = false) and ((common0).en[37]
= false) and ((common0).en[42] = false) and ((common0).en[43] = false) and ((common0).en[44]
= false);
task (common0).exec[24] := true;
nextstate schedule_navigator_navsteering_points;

provided (common0).en[25] and ((common0).en[8] = false) and ((common0).en[31] = false)
and ((common0).en[32] = false) and ((common0).en[33] = false) and ((common0).en[34] =
false) and ((common0).en[35] = false) and ((common0).en[36] = false) and ((common0).en[42]
= false) and ((common0).en[43] = false) and ((common0).en[44] = false);
task (common0).exec[25] := true;
nextstate schedule_phase_manager;

provided (common0).en[27] and ((common0).en[3] = false) and ((common0).en[8] = false)
and ((common0).en[25] = false) and ((common0).en[31] = false) and ((common0).en[32] =
false) and ((common0).en[33] = false) and ((common0).en[34] = false) and ((common0).en[35]
= false) and ((common0).en[36] = false) and ((common0).en[42] = false) and ((common0).en[43]
= false) and ((common0).en[44] = false);
task (common0).exec[27] := true;
nextstate schedule_pilot_waypoints;

provided (common0).en[31];
task (common0).exec[31] := true;
nextstate schedule_radio1_device;

provided (common0).en[32] and ((common0).en[31] = false);
task (common0).exec[32] := true;
nextstate schedule_radio1_logical_device;

provided (common0).en[33] and ((common0).en[31] = false) and ((common0).en[32] = false);
task (common0).exec[33] := true;
nextstate schedule_radio1_logical_display;

provided (common0).en[34] and ((common0).en[31] = false) and ((common0).en[32] = false)
and ((common0).en[33] = false);
task (common0).exec[34] := true;
nextstate schedule_radio2_device;
```

```
provided (common0).en[35] and ((common0).en[31] = false) and ((common0).en[32] = false)
and ((common0).en[33] = false) and ((common0).en[34] = false);
task (common0).exec[35] := true;
nextstate schedule_radio2_logical_device;

provided (common0).en[36] and ((common0).en[31] = false) and ((common0).en[32] = false)
and ((common0).en[33] = false) and ((common0).en[34] = false) and ((common0).en[35] =
false);
task (common0).exec[36] := true;
nextstate schedule_radio2_logical_display;

provided (common0).en[37] and ((common0).en[3] = false) and ((common0).en[8] = false)
and ((common0).en[25] = false) and ((common0).en[27] = false) and ((common0).en[31] =
false) and ((common0).en[32] = false) and ((common0).en[33] = false) and ((common0).en[34]
= false) and ((common0).en[35] = false) and ((common0).en[36] = false) and ((common0).en[42]
= false) and ((common0).en[43] = false) and ((common0).en[44] = false);
task (common0).exec[37] := true;
nextstate schedule_routes;

provided (common0).en[42] and ((common0).en[8] = false) and ((common0).en[31] = false)
and ((common0).en[32] = false) and ((common0).en[33] = false) and ((common0).en[34] =
false) and ((common0).en[35] = false) and ((common0).en[36] = false) and ((common0).en[44]
= false);
task (common0).exec[42] := true;
nextstate schedule_tactical_logical_device;

provided (common0).en[43] and ((common0).en[8] = false) and ((common0).en[31] = false)
and ((common0).en[32] = false) and ((common0).en[33] = false) and ((common0).en[34] =
false) and ((common0).en[35] = false) and ((common0).en[36] = false) and ((common0).en[42]
= false) and ((common0).en[44] = false);
task (common0).exec[43] := true;
nextstate schedule_tactical_mode;

provided (common0).en[44] and ((common0).en[8] = false) and ((common0).en[31] = false)
and ((common0).en[32] = false) and ((common0).en[33] = false) and ((common0).en[34] =
false) and ((common0).en[35] = false) and ((common0).en[36] = false);
task (common0).exec[44] := true;
nextstate schedule_tactical_physical_device_5Hz;

endstate;

state schedule_airframe;
deadline eager;
provided (common0).en[8] or (common0).en[25] or (common0).en[31] or (common0).en[32] or
(common0).en[33] or (common0).en[34] or (common0).en[35] or (common0).en[36] or (common0).en[42]
or (common0).en[43] or (common0).en[44];
when (task_airframe0).ce = 0;
task (common0).exec[3] := false;
nextstate initial;
provided (common0).en[3] = false;
task (common0).exec[3] := false;
nextstate initial;

endstate;
```

```
state schedule_device_5Hz;
deadline eager;
provided (common0).en[31] or (common0).en[32] or (common0).en[33] or (common0).en[34]
or (common0).en[35] or (common0).en[36];
when (task_device_5Hz0).ce = 0;
task (common0).exec[8] := false;
nextstate initial;
provided (common0).en[8] = false;
task (common0).exec[8] := false;
nextstate initial;

endstate;

state schedule_nav_steering;
deadline eager;
provided (common0).en[3] or (common0).en[8] or (common0).en[24] or (common0).en[25] or
(common0).en[27] or (common0).en[31] or (common0).en[32] or (common0).en[33] or (common0).en[34]
or (common0).en[35] or (common0).en[36] or (common0).en[37] or (common0).en[42] or (common0).en[43]
or (common0).en[44];
when (task_nav_steering0).ce = 0;
task (common0).exec[23] := false;
nextstate initial;
provided (common0).en[23] = false;
task (common0).exec[23] := false;
nextstate initial;

endstate;

state schedule_navigator_navsteering_points;
deadline eager;
provided (common0).en[3] or (common0).en[8] or (common0).en[25] or (common0).en[27] or
(common0).en[31] or (common0).en[32] or (common0).en[33] or (common0).en[34] or (common0).en[35]
or (common0).en[36] or (common0).en[37] or (common0).en[42] or (common0).en[43] or (common0).en[44];
when (task_navigator_navsteering_points0).ce = 0;
task (common0).exec[24] := false;
nextstate initial;
provided (common0).en[24] = false;
task (common0).exec[24] := false;
nextstate initial;

endstate;

state schedule_phase_manager;
deadline eager;
provided (common0).en[8] or (common0).en[31] or (common0).en[32] or (common0).en[33] or
(common0).en[34] or (common0).en[35] or (common0).en[36] or (common0).en[42] or (common0).en[43]
or (common0).en[44];
when (task_phase_manager0).ce = 0;
task (common0).exec[25] := false;
nextstate initial;
provided (common0).en[25] = false;
task (common0).exec[25] := false;
nextstate initial;

endstate;
```

```
state schedule_pilot_waypoints;
deadline eager;
provided (common0).en[3] or (common0).en[8] or (common0).en[25] or (common0).en[31] or
(common0).en[32] or (common0).en[33] or (common0).en[34] or (common0).en[35] or (common0).en[36]
or (common0).en[42] or (common0).en[43] or (common0).en[44];
when (task_pilot_waypoints0).ce = 0;
task (common0).exec[27] := false;
nextstate initial;
provided (common0).en[27] = false;
task (common0).exec[27] := false;
nextstate initial;

endstate;

state schedule_radio1_device;
provided (common0).en[31] = false;
task (common0).exec[31] := false;
nextstate initial;

endstate;

state schedule_radio1_logical_device;
deadline eager;
provided (common0).en[31];
when (task_radio1_logical_device0).ce = 0;
task (common0).exec[32] := false;
nextstate initial;
provided (common0).en[32] = false;
task (common0).exec[32] := false;
nextstate initial;

endstate;

state schedule_radio1_logical_display;
deadline eager;
provided (common0).en[31] or (common0).en[32];
when (task_radio1_logical_display0).ce = 0;
task (common0).exec[33] := false;
nextstate initial;
provided (common0).en[33] = false;
task (common0).exec[33] := false;
nextstate initial;

endstate;

state schedule_radio2_device;
deadline eager;
provided (common0).en[31] or (common0).en[32] or (common0).en[33];
when (task_radio2_device0).ce = 0;
task (common0).exec[34] := false;
nextstate initial;
provided (common0).en[34] = false;
task (common0).exec[34] := false;
nextstate initial;

endstate;
```

```
state schedule_radio2_logical_device;
deadline eager;
provided (common0).en[31] or (common0).en[32] or (common0).en[33] or (common0).en[34];
when (task_radio2_logical_device0).ce = 0;
task (common0).exec[35] := false;
nextstate initial;
provided (common0).en[35] = false;
task (common0).exec[35] := false;
nextstate initial;

endstate;

state schedule_radio2_logical_display;
deadline eager;
provided (common0).en[31] or (common0).en[32] or (common0).en[33] or (common0).en[34]
or (common0).en[35];
when (task_radio2_logical_display0).ce = 0;
task (common0).exec[36] := false;
nextstate initial;
provided (common0).en[36] = false;
task (common0).exec[36] := false;
nextstate initial;

endstate;

state schedule_routes;
deadline eager;
provided (common0).en[3] or (common0).en[8] or (common0).en[25] or (common0).en[27] or
(common0).en[31] or (common0).en[32] or (common0).en[33] or (common0).en[34] or (common0).en[35]
or (common0).en[36] or (common0).en[42] or (common0).en[43] or (common0).en[44];
when (task_routes0).ce = 0;
task (common0).exec[37] := false;
nextstate initial;
provided (common0).en[37] = false;
task (common0).exec[37] := false;
nextstate initial;

endstate;

state schedule_tactical_logical_device;
deadline eager;
provided (common0).en[8] or (common0).en[31] or (common0).en[32] or (common0).en[33] or
(common0).en[34] or (common0).en[35] or (common0).en[36] or (common0).en[44];
when (task_tactical_logical_device0).ce = 0;
task (common0).exec[42] := false;
nextstate initial;
provided (common0).en[42] = false;
task (common0).exec[42] := false;
nextstate initial;

endstate;

state schedule_tactical_mode;
deadline eager;
provided (common0).en[8] or (common0).en[31] or (common0).en[32] or (common0).en[33] or
```

```
(common0).en[34] or (common0).en[35] or (common0).en[36] or (common0).en[42] or (common0).en[44];
when (task_tactical_mode0).ce = 0;
task (common0).exec[43] := false;
nextstate initial;
provided (common0).en[43] = false;
task (common0).exec[43] := false;
nextstate initial;

endstate;

state schedule_tactical_physical_device_5Hz;
deadline eager;
provided (common0).en[8] or (common0).en[31] or (common0).en[32] or (common0).en[33] or
(common0).en[34] or (common0).en[35] or (common0).en[36];
when (task_tactical_physical_device_5Hz0).ce = 0;
task (common0).exec[44] := false;
nextstate initial;
provided (common0).en[44] = false;
task (common0).exec[44] := false;
nextstate initial;

endstate;

endprocess;

endsystem;

/*
* Observer
*
*/

pure observer safety;

state idle #start ;
deadline eager;

provided (task_af_monitor0) instate wait;
when (task_af_monitor0).cd >= dl0;
nextstate error;
provided (task_af_monitor0) instate run;
when (task_af_monitor0).cd >= dl0;
nextstate error;

provided (task_af_state_logical_device0) instate wait;
when (task_af_state_logical_device0).cd >= dl1;
nextstate error;
provided (task_af_state_logical_device0) instate run;
when (task_af_state_logical_device0).cd >= dl1;
nextstate error;

provided (task_af_state_logical_display0) instate wait;
when (task_af_state_logical_display0).cd >= dl2;
nextstate error;
provided (task_af_state_logical_display0) instate run;
when (task_af_state_logical_display0).cd >= dl2;
```

```
nextstate error;

provided (task_airframe0) instate wait;
when (task_airframe0).cd >= dl3;
nextstate error;
provided (task_airframe0) instate run;
when (task_airframe0).cd >= dl3;
nextstate error;

provided (task_aths_device0) instate wait;
when (task_aths_device0).cd >= dl4;
nextstate error;
provided (task_aths_device0) instate run;
when (task_aths_device0).cd >= dl4;
nextstate error;

provided (task_aths_logical_device0) instate wait;
when (task_aths_logical_device0).cd >= dl5;
nextstate error;
provided (task_aths_logical_device0) instate run;
when (task_aths_logical_device0).cd >= dl5;
nextstate error;

provided (task_aths_logical_display0) instate wait;
when (task_aths_logical_display0).cd >= dl6;
nextstate error;
provided (task_aths_logical_display0) instate run;
when (task_aths_logical_display0).cd >= dl6;
nextstate error;

provided (task_cnidc_device0) instate wait;
when (task_cnidc_device0).cd >= dl7;
nextstate error;
provided (task_cnidc_device0) instate run;
when (task_cnidc_device0).cd >= dl7;
nextstate error;

provided (task_device_5Hz0) instate wait;
when (task_device_5Hz0).cd >= dl8;
nextstate error;
provided (task_device_5Hz0) instate run;
when (task_device_5Hz0).cd >= dl8;
nextstate error;

provided (task_display_device0) instate wait;
when (task_display_device0).cd >= dl9;
nextstate error;
provided (task_display_device0) instate run;
when (task_display_device0).cd >= dl9;
nextstate error;

provided (task_dmt_device0) instate wait;
when (task_dmt_device0).cd >= dl10;
nextstate error;
provided (task_dmt_device0) instate run;
when (task_dmt_device0).cd >= dl10;
```

```
nextstate error;

provided (task_dmt_logical_device0) instate wait;
when (task_dmt_logical_device0).cd >= dl11;
nextstate error;
provided (task_dmt_logical_device0) instate run;
when (task_dmt_logical_device0).cd >= dl11;
nextstate error;

provided (task_dmt_logical_display0) instate wait;
when (task_dmt_logical_display0).cd >= dl12;
nextstate error;
provided (task_dmt_logical_display0) instate run;
when (task_dmt_logical_display0).cd >= dl12;
nextstate error;

provided (task_dvms_device0) instate wait;
when (task_dvms_device0).cd >= dl13;
nextstate error;
provided (task_dvms_device0) instate run;
when (task_dvms_device0).cd >= dl13;
nextstate error;

provided (task_dvms_logical_device0) instate wait;
when (task_dvms_logical_device0).cd >= dl14;
nextstate error;
provided (task_dvms_logical_device0) instate run;
when (task_dvms_logical_device0).cd >= dl14;
nextstate error;

provided (task_dvms_logical_display0) instate wait;
when (task_dvms_logical_display0).cd >= dl15;
nextstate error;
provided (task_dvms_logical_display0) instate run;
when (task_dvms_logical_display0).cd >= dl15;
nextstate error;

provided (task_ems_device0) instate wait;
when (task_ems_device0).cd >= dl16;
nextstate error;
provided (task_ems_device0) instate run;
when (task_ems_device0).cd >= dl16;
nextstate error;

provided (task_ems_logical_device0) instate wait;
when (task_ems_logical_device0).cd >= dl17;
nextstate error;
provided (task_ems_logical_device0) instate run;
when (task_ems_logical_device0).cd >= dl17;
nextstate error;

provided (task_ems_logical_display0) instate wait;
when (task_ems_logical_display0).cd >= dl18;
nextstate error;
provided (task_ems_logical_display0) instate run;
when (task_ems_logical_display0).cd >= dl18;
```

```
nextstate error;

provided (task_miu_device0) instate wait;
when (task_miu_device0).cd >= dl19;
nextstate error;
provided (task_miu_device0) instate run;
when (task_miu_device0).cd >= dl19;
nextstate error;

provided (task_miu_logical_device0) instate wait;
when (task_miu_logical_device0).cd >= dl20;
nextstate error;
provided (task_miu_logical_device0) instate run;
when (task_miu_logical_device0).cd >= dl20;
nextstate error;

provided (task_miu_logical_display0) instate wait;
when (task_miu_logical_display0).cd >= dl21;
nextstate error;
provided (task_miu_logical_display0) instate run;
when (task_miu_logical_display0).cd >= dl21;
nextstate error;

provided (task_nav_display0) instate wait;
when (task_nav_display0).cd >= dl22;
nextstate error;
provided (task_nav_display0) instate run;
when (task_nav_display0).cd >= dl22;
nextstate error;

provided (task_nav_steering0) instate wait;
when (task_nav_steering0).cd >= dl23;
nextstate error;
provided (task_nav_steering0) instate run;
when (task_nav_steering0).cd >= dl23;
nextstate error;

provided (task_navigator_navsteering_points0) instate wait;
when (task_navigator_navsteering_points0).cd >= dl24;
nextstate error;
provided (task_navigator_navsteering_points0) instate run;
when (task_navigator_navsteering_points0).cd >= dl24;
nextstate error;

provided (task_phase_manager0) instate wait;
when (task_phase_manager0).cd >= dl25;
nextstate error;
provided (task_phase_manager0) instate run;
when (task_phase_manager0).cd >= dl25;
nextstate error;

provided (task_pilot_control0) instate wait;
when (task_pilot_control0).cd >= dl26;
nextstate error;
provided (task_pilot_control0) instate run;
when (task_pilot_control0).cd >= dl26;
```

```
nextstate error;

provided (task_pilot_waypoints0) instate wait;
when (task_pilot_waypoints0).cd >= dl27;
nextstate error;
provided (task_pilot_waypoints0) instate run;
when (task_pilot_waypoints0).cd >= dl27;
nextstate error;

provided (task_radar_device0) instate wait;
when (task_radar_device0).cd >= dl28;
nextstate error;
provided (task_radar_device0) instate run;
when (task_radar_device0).cd >= dl28;
nextstate error;

provided (task_radar_logical_device0) instate wait;
when (task_radar_logical_device0).cd >= dl29;
nextstate error;
provided (task_radar_logical_device0) instate run;
when (task_radar_logical_device0).cd >= dl29;
nextstate error;

provided (task_radar_logical_display0) instate wait;
when (task_radar_logical_display0).cd >= dl30;
nextstate error;
provided (task_radar_logical_display0) instate run;
when (task_radar_logical_display0).cd >= dl30;
nextstate error;

provided (task_radio1_device0) instate wait;
when (task_radio1_device0).cd >= dl31;
nextstate error;
provided (task_radio1_device0) instate run;
when (task_radio1_device0).cd >= dl31;
nextstate error;

provided (task_radio1_logical_device0) instate wait;
when (task_radio1_logical_device0).cd >= dl32;
nextstate error;
provided (task_radio1_logical_device0) instate run;
when (task_radio1_logical_device0).cd >= dl32;
nextstate error;

provided (task_radio1_logical_display0) instate wait;
when (task_radio1_logical_display0).cd >= dl33;
nextstate error;
provided (task_radio1_logical_display0) instate run;
when (task_radio1_logical_display0).cd >= dl33;
nextstate error;

provided (task_radio2_device0) instate wait;
when (task_radio2_device0).cd >= dl34;
nextstate error;
provided (task_radio2_device0) instate run;
when (task_radio2_device0).cd >= dl34;
```

```
nextstate error;

provided (task_radio2_logical_device0) instate wait;
when (task_radio2_logical_device0).cd >= dl35;
nextstate error;
provided (task_radio2_logical_device0) instate run;
when (task_radio2_logical_device0).cd >= dl35;
nextstate error;

provided (task_radio2_logical_display0) instate wait;
when (task_radio2_logical_display0).cd >= dl36;
nextstate error;
provided (task_radio2_logical_display0) instate run;
when (task_radio2_logical_display0).cd >= dl36;
nextstate error;

provided (task_routes0) instate wait;
when (task_routes0).cd >= dl37;
nextstate error;
provided (task_routes0) instate run;
when (task_routes0).cd >= dl37;
nextstate error;

provided (task_rwr_device0) instate wait;
when (task_rwr_device0).cd >= dl38;
nextstate error;
provided (task_rwr_device0) instate run;
when (task_rwr_device0).cd >= dl38;
nextstate error;

provided (task_rwr_logical_device0) instate wait;
when (task_rwr_logical_device0).cd >= dl39;
nextstate error;
provided (task_rwr_logical_device0) instate run;
when (task_rwr_logical_device0).cd >= dl39;
nextstate error;

provided (task_rwr_logical_display0) instate wait;
when (task_rwr_logical_display0).cd >= dl40;
nextstate error;
provided (task_rwr_logical_display0) instate run;
when (task_rwr_logical_display0).cd >= dl40;
nextstate error;

provided (task_saahs_device0) instate wait;
when (task_saahs_device0).cd >= dl41;
nextstate error;
provided (task_saahs_device0) instate run;
when (task_saahs_device0).cd >= dl41;
nextstate error;

provided (task_tactical_logical_device0) instate wait;
when (task_tactical_logical_device0).cd >= dl42;
nextstate error;
provided (task_tactical_logical_device0) instate run;
when (task_tactical_logical_device0).cd >= dl42;
```

```
nextstate error;

provided (task_tactical_mode0) instate wait;
when (task_tactical_mode0).cd >= dl43;
nextstate error;
provided (task_tactical_mode0) instate run;
when (task_tactical_mode0).cd >= dl43;
nextstate error;

provided (task_tactical_physical_device_5Hz0) instate wait;
when (task_tactical_physical_device_5Hz0).cd >= dl44;
nextstate error;
provided (task_tactical_physical_device_5Hz0) instate run;
when (task_tactical_physical_device_5Hz0).cd >= dl44;
nextstate error;

provided (task_tactical_steering0) instate wait;
when (task_tactical_steering0).cd >= dl45;
nextstate error;
provided (task_tactical_steering0) instate run;
when (task_tactical_steering0).cd >= dl45;
nextstate error;

provided (task_tacts_device0) instate wait;
when (task_tacts_device0).cd >= dl46;
nextstate error;
provided (task_tacts_device0) instate run;
when (task_tacts_device0).cd >= dl46;
nextstate error;

provided (task_tacts_logical_device0) instate wait;
when (task_tacts_logical_device0).cd >= dl47;
nextstate error;
provided (task_tacts_logical_device0) instate run;
when (task_tacts_logical_device0).cd >= dl47;
nextstate error;

provided (task_tacts_logical_display0) instate wait;
when (task_tacts_logical_display0).cd >= dl48;
nextstate error;
provided (task_tacts_logical_display0) instate run;
when (task_tacts_logical_display0).cd >= dl48;
nextstate error;

provided (task_wmc_device0) instate wait;
when (task_wmc_device0).cd >= dl49;
nextstate error;
provided (task_wmc_device0) instate run;
when (task_wmc_device0).cd >= dl49;
nextstate error;

provided (task_wmc_logical_device0) instate wait;
when (task_wmc_logical_device0).cd >= dl50;
nextstate error;
provided (task_wmc_logical_device0) instate run;
when (task_wmc_logical_device0).cd >= dl50;
```

```
nextstate error;

provided (task_wmc_logical_display0) instate wait;
when (task_wmc_logical_display0).cd >= dl51;
nextstate error;
provided (task_wmc_logical_display0) instate run;
when (task_wmc_logical_display0).cd >= dl51;
nextstate error;

endstate;

state error #error ;
endstate;

endobserver;
```