Developing a Formal Specification for the Mission System of a Maritime Surveillance Aircraft

Laure Petrucci
LSV, CNRS UMR 8643
ENS de Cachan
61, avenue du Président Wilson
94235 CACHAN Cedex, France
petrucci@lsv.ens-cachan.fr

Jonathan Billington
Computer Systems Engineering Centre
School of Elec. and Information Engineering
University of South Australia
MAWSON LAKES, SA 5095, Australia
jonathan.billington@unisa.edu.au

Lars M. Kristensen*
Department of Computer Science
University of Aarhus
IT-parken, Aabogade 34
8200 AARHUS N, Denmark
kris@daimi.au.dk

Zahid H. Qureshi
System Sciences Laboratory
Defence Science and Technology Organisation
EDINBURGH, SA 5111, Australia
zahid.qureshi@dsto.defence.gov.au

Abstract

The mission system of an aircraft is a complex real-time distributed system consisting of a mission control computer, different kinds of devices interconnected by a number of serial data buses. The complexity and real-time requirements of mission systems have motivated research into the application of formal techniques to investigate and predict the effects of upgrades on mission system behaviour. This paper reports on a joint research project between the University of South Australia and Australia's Defence Science and Technology Organisation. In previous work we modelled a generic avionics mission system with Coloured Petri Nets and analysed the model using state spaces. Here, we describe how this model was refined and modified to obtain a Coloured Petri Net model for the AP-3C Orion maritime surveillance aircraft.

1. Introduction

Mission system upgrades to aircraft platforms are labour intensive and require significant capital expenditure by national Defence Forcesmainly because: changes to system components normally require substantial redesign and testing; their integration can lead to unforeseen and undesirable behaviour due to complex interactions between components. The mission system [16] is critical for the successful deployment and operation of military aircraft. It is a complex real-time system [9], responsible for the tactical control of devices such as radars and sensors of the aircraft during a mission. The tasks performed by the mission system include data collection from sensors, fusion of collected data, display of information to pilots, and controlling devices in response to inputs from the aircraft crew. To ensure that the system is working properly, the tasks must be scheduled in a way that guarantees that hard deadlines are met. This requirement is critical to the success of a mission. Thus major concerns when upgrading and maintaining mission systems are the scheduling of tasks and the impact of the delays associated with data transfer across the bus connecting the mission control computer and the devices.

In [12, 13], Timed Coloured Petri Nets (CP-nets or CPNs) [7, 8] were used to model a generic airborne mission system [10] with the purpose of investigating scheduling. Design/CPN [5] was then used to find a schedule. Scheduling of tasks was treated as a reachability problem, using a state space search. The use of state space methods for

*Supported by the Danish Natural Science Research Council

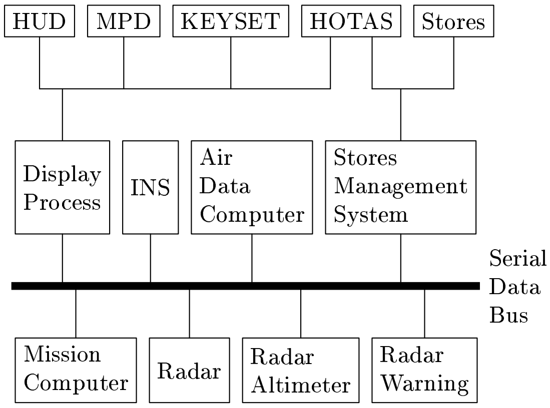
scheduling was motivated by recent work [2, 3, 6, 17] in the area of timed automata [1], where state space methods have been used to synthesize schedules for the control of real-time systems. The advantage of formal modelling and state space methods in this setting is that the same model of the system can be used to analyse scheduling as well as other functional and performance properties. Hence, it provides an integrated approach for predicting system behaviour.

In this paper, we show how a specification for the AP-3C Orion aircraft, used for maritime surveillance, could be derived from the generic model. The CPN model was created by a team of engineers and researchers from RLM Systems [14], Australia's Defence Science and Technology Organisation (DSTO) [15], and the Computer Systems Engineering Centre (CSEC) at the University of South Australia[4]. The purpose of constructing the CPN model of the AP-3C mission system was to investigate whether the modelling framework for the generic mission system [12, 13] could be applied to develop a CPN model of the AMS of a real aircraft. RLM Systems and DSTO considered a formal and executable CPN specification a valuable supplement to existing design documentation of the AP-3C aircraft. A long-term goal for DSTO and RLM Systems is to integrate the CPN model into their evaluation environment for the AP-3C aircraft. The focus for the part of the project reported on in this paper is therefore modelling and specification, rather than state space analysis.

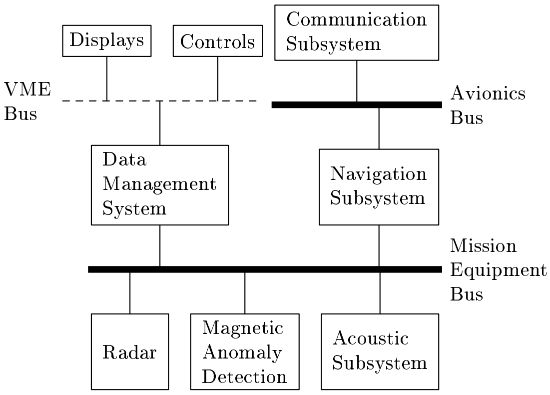
This paper is organised as follows. Sect. 2 presents the mission system architecture for the AP-3C platform and compares it with the generic architecture. Sect. 3 compares the overall structure of the CPN model of the generic AMS to the AP-3C mission system model. Sect. 4 presents how subsystems of the AP-3C mission system were modelled. Sect. 5 details the I/O processing and messages transmission. Finally, in Sect. 6 we provide conclusions and discuss future work. We assume that the reader is familiar with the basic ideas of the CPN modelling language [7, 8].

2. Mission System Architectures

An Airborne Mission System (AMS) architecture is characterized by a set of sub-systems or components interconnected via serial data busses. The architecture of the generic avionics mission system [10] is depicted in Fig. 1(a)[1] [13]. It consists of a mission control computer, a serial data bus, and external components such as the navigation system, radars and sensors. The various subsystems and displays are controlled by executing a set of tasks on the mission control computer (Mission Computer in Fig. 1(a)).



(a) Generic AMS architecture.



(b) The AP-3C AMS architecture.

Figure 1. The AMS architectures.

Fig. 1(b) shows a high-level architecture block diagram of the AP-3C AMS, consisting of a *Data Management System* (DMS) which communicates with several subsystems, such as *Radar*, *Magnetic Anomaly Detection* (MAD), and the *Acoustic Subsystem* via the *Mission Equipment Bus* (MEB). The MEB is a dual MIL-STD-1553B [11] serial data bus. The DMS is also connected to the *Navigation Subsystem* of the aircraft. Another 1553B bus, the *Avionics Bus*, connects the *Navi-*

[1]The displays and controls include the *HUD* (Heads-Up Display), *MPD* (Multi-Purpose Display), *KEYSET* (keyboard for crew) and *HOTAS* (Hands-On Throttle And Stick). Among the devices is the *INS* (Inertial Navigation System).

gation Subsystem and the *Communication Subsystem*. The *Displays* and *Controls* allow the crew to monitor and control the aircraft and are connected to the DMS by a VME bus. The DMS of the AP-3C plays the same role as the mission control computer in the generic mission system.

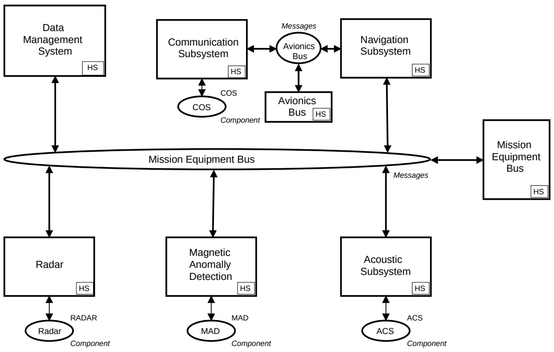


Figure 2. The *AMS* page.

From the two architectures depicted in Fig. 1(a) (generic) and 1(b) it follows that although the components in the two systems are basically the same, the structure is quite different. Firstly, there are two 1553B busses in the AP-3C instead of one in the generic AMS. This is easy to handle in our model as they are identical, allowing us to use two instances of the same bus representation. We just need to connect the appropriate components to each bus instance to ensure that the model correctly reflects the topology of the system.

Secondly, a display processor is attached to the serial data bus in the generic mission system, while it is directly connected to the data management system in the AP-3C, through a VME bus. Based on the experience of the domain experts from DSTO and RLM Systems, it was decided not to model the VME bus, as the VME bus is a high-bandwidth bus carrying little traffic relative to the 1553B bus. It is therefore unlikely that it would become the bottleneck in the system. Hence, we consider that omitting the VME bus from the model will not compromise the analysis results.

Finally, we need to consider two sorts of subsystems (or *devices*) in the AP-3C mission system, rather than one. Because of the level of abstraction chosen for the model, we call subsystems or devices that have just a single connection to a bus, a *simple device*. These are *Radar*, *Magnetic Anomaly Detection*, the *Acoustic Subsystem* and the *Communication Subsystem*. We need a more complex model of the *Navigation Subsystem* because it is connected to both busses. Therefore, the simple devices can be modelled in the same way as the devices of the generic mission system (the *Generic Sensor*), while the navigation subsystem requires an enhancement to this model.

3. Modelling AMS Architectures

The AMS systems under consideration are modelled by means of Hierarchical Timed Coloured Petri Nets [7, 8] (CP-nets or CPNs). CP-nets allow the model to be split into a number of hierarchically related *pages* (modules) via the concepts of *substitution transitions* and *subpages*. A page on one level may have a number of substitution transitions, typically representing a subsystem or some compound behaviour. A substitution transition has an associated subpage modelling the behaviour represented by the substitution transition in more detail. A subpage may again contain substitution transitions giving rise to a hierarchical organization of the model.

Fig. 2 depicts the page *AMS* from the AP-3C model. This page provides a high-level architectural view of the AP-3C AMS similar to the informal drawing in Fig. 1(b). All the transitions (rectangles) are substitution transitions, indicated by the hierarchical substitution tag (HS) in their lower right corner. The substitution transition *Data Management System* represents the main part of the system. The *Communication Subsystem*, *Navigation Subsystem*, *Acoustic Subsystem*, *Magnetic Anomaly Detection* and *Radar* correspond to the different AMS subsystems. The other two substitution transitions, *Mission Equipment Bus* and *Avionics Bus* are used to model data transfer across the MIL-STD-1553B serial data buses. Places *Mission Equipment Bus* and *Avionics Bus* represent the interfaces for each bus. Finally, the other places (e.g. *ACS*) allow subsystems (such as the acoustics subsystem) to be identified.

The *hierarchy page* of the generic AMS model is shown in Fig. 3. This page shows the overall organization of the pages constituting the CPN model. Each page of the CPN model is represented by a node on the hierarchy page. An arrow from one node to another node indicates that the latter is a subpage (submodule) of the former. It can be seen that the CPN model of the generic AMS consists of 17 pages. The generic AMS consists of a *MCC* (Mission Control Computer), a *Serial Data Bus*, *ControlsDisplays*, and *Stores*. These correspond to the main components of the generic AMS as shown in Fig. 1(a).

The hierarchy page of the AP-3C CPN model

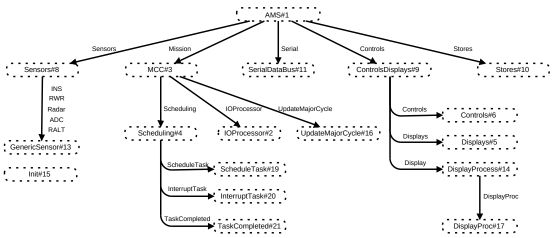


Figure 3. Hierarchy page for the generic AMS model.

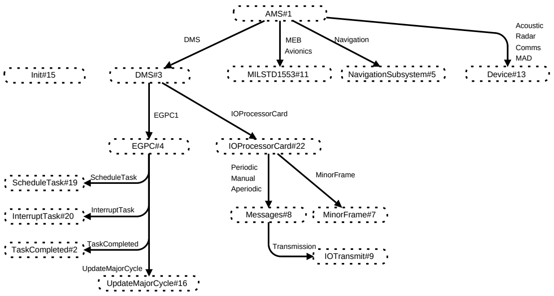


Figure 4. Hierarchy for the AP-3C model.

is depicted in Fig. 4. The AP-3C model consists of 14 pages. The AP-3C *Airborne Mission System* (AMS) consists of a *Data Management System* (DMS) comprising an *Enhanced General Purpose Controller* responsible for executing a set of tasks, and an *IOProcessor Card*. The *Data Management System* communicates with several simple *Devices*, and with the *Navigation Subsystem* of the aircraft via a dual *MIL-STD-1553B* mission equipment bus (MEB). There are four *instances* of the *Device* page corresponding to the four simple devices in the system. This can be seen from the four labels on the arc from page *AMS* to page *Device*. The labels correspond to names of the substitution transitions at the source page of the arc. This means that each simple device is modelled by the page *Device* and at the abstraction level of the CPN model, all simple devices behave the same way. The *Avionics Bus* sits between the *Navigation Subsystem* and the *Communication Subsystem*. Since the MEB and the avionics bus are identical, there are two instances of the *MILSTD1553* page, one corresponding to the MEB, and the other to the avionics bus.

A parallel should be drawn between the generic model hierarchy in Fig. 3 and the one of the AP-3C (see Fig. 4). The four main differences between the CPN models of the generic AMS and the AP-3C are reflected in the hierarchy pages as follows:

- The *Stores* and the *ControlsDisplays* pages and subpages have been removed from the model as we are initially concerned with scheduling problems associated with the DMS not related to the displays and controls.

- The *Sensors* and *Generic Sensor* pages from the generic AMS model have been combined and then split into two: the simple devices and the navigation subsystem.

- The CPN model of the AP-3C AMS contains a refined model of the input/output processing on the mission control computer. In the generic CPN model, I/O processing was modelled by page *IOProcessor*. In the AP-3C model, it is modelled by page *IOProcessorCard* and its three subpages. The *Scheduling* page of the generic model becomes the *EGPC* page for the AP-3C.

- The timing regarding task execution has been moved from the mission control computer level to the EGPC level. Hence, page *UpdateMajorCycle* is now a subpage of the *EGPC* page.

The differences between the AP-3C and the generic AMS architectures are easily recognised by examining the CPN models' hierarchy pages. The transformation of the generic model into the AP-3C model was greatly facilitated by its initial hierarchical design. The transformation mainly consists of: re-arranging the hierarchy by moving some pages; creating new ones when refinement is required; and deleting pages not relevant to the specific architecture or the purpose of the model.

One purpose of the CPN model is to formally specify the transmission of messages between subsystems across the mission equipment and avionics busses. As usual we model the messages being transferred as tokens in the CPN model. When a subsystem transmits a message across the mission equipment bus to another subsystem, it will put a token on place *Mission Equipment Bus*. The subpage of the substitution transition *Mission Equipment Bus* will then model the details in transferring the message. Eventually the message will have been transmitted and the destination subsystem will consume the token representing the message from place *Mission Equipment Bus*. To model this data transfer in a flexible way that makes it easy to add/remove components, a general addressing scheme was developed as part of the CPN model of the generic AMS.

Fig. 5 shows selected colour set declarations used to implement the addressing scheme for the AP-3C, and the modelling of messages transferred. The *Component* colour set is a union colour set corresponding to the components of the AMS. The *MCCTask* colour set is used to specify the tasks executing on the DMS. The addressing scheme makes it possible to identify the specific tasks in the DMS. This is required since tasks will be blocked waiting for responses from subsystems when messages are being sent across the buses. Hence, it is necessary to be able to identify the task that is to receive a certain message.

The *SDBCommand* colour set is used to specify the sender, receiver and size of a message. It is important to notice that we do not model the content of messages. We are only interested in modelling that a message is being transmitted, and the timing related to this message transfer. The colour set *SDBMsg* is used for modelling the messages (control signals) at the local interface between the subsystems and the bus.

The colour set *Messages*, colour set of the place *Mission Equipment Bus* in Fig. 2, is used to represent messages in transit. When a subsystem puts a message on the bus, the first component of the tuple representing the message will be *BUS* to signal that the message is currently at the bus interface. The bus will then transfer the message as modelled by the subpage of the substitution transition *Mission Equipment Bus*. This will produce a token on place *Mission Equipment Bus* where the first component of the tuple identifies the destination subsystem. This subsystem may then receive the message by consuming the token.

```
color Component = union
       DMS : MCCTask +
       IOP + BUS + RADAR + MAD +
       ACS + NAS + COS;
color SDBCommand = record src : Component *
                          dest : Component *
                          spec : Int;
color SDBMsg = union  SDBCOM : SDBCommand +
               IOstart : Int + IOcomplete;
color Messages = product Component * SDBMsg;
```

Figure 5. Selected colour set declarations for component addressing.

4. Modelling AMS Subsystems

We now consider the modelling of the individual subsystems of the AP-3C platform.

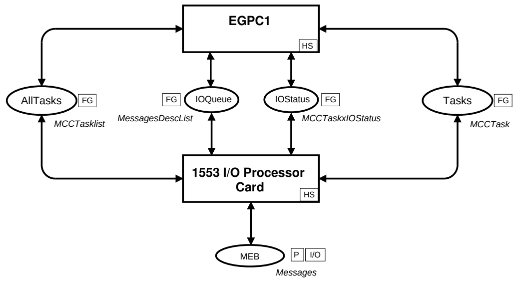


Figure 6. The *DMS* page.

4.1. Data Management System

The Data Management System (DMS) is the core of the airborne mission system. Fig. 6 depicts the most abstract part modelling the data management system, the *DMS* page. It has two substitution transitions: *EGPC1* represents the *Enhanced General Purpose Controller* on which the tasks execute, and the *1553B I/O Processor Card* handles all the input and output related to tasks. The *IO Processor Card* is described in detail in Sect. 5. The AP-3C aircraft uses up to 4 EGPCs. Our model can easily cater for this by just including the required number of EGPC substitution transitions (e.g. *EGPC1* to *EGPC4*). Thus, several instances of the *EGPC* page may be used concurrently.

Input socket places *Tasks* and *AllTasks* store the tasks to be executed for the mission. Each task is represented as a token of colour set *MCCTask* (see Fig. 7) on the place *Tasks*. Place *AllTasks* contains a list of all tasks. It is used to access all tasks to determine which one will be executed next, according to the task priority mechanism on the EPGC. The *Tasks* place ensures that if there is a choice of the next task to execute, the CPN model represents all possible choices. This might seem redundant information, but having both places helps in making an non-deterministic choice among tasks and priority calculus.

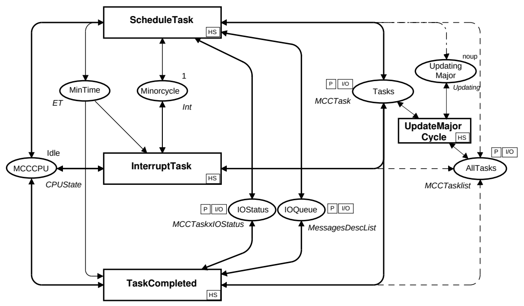


Figure 8. The *EGPC* page.

```
color Int = int;
color Bool = bool;
color MCCTaskName = string;
color TaskStatus = with TaskIdle            | TaskDataRequest |
                        TaskWaitingWithData | TaskExecuting   | TaskResponse;
color BackgroundTask = record
         name : MCCTaskName *
         size : Int *         (* size of task in time units *)
         left : Int *         (* time units left of the job *)
         exeinmajor : Bool *  (* if executed in the major cycle *)
         priority : Int *     (* priority among backg. tasks *)
         status : TaskStatus; (* for priority computation *)
color RategroupTask = record
         name : MCCTaskName *
         rate : Int *         (* time units between executions *)
         next : Int *         (* next minor cycle to be scheduled *)
         size : Int *         (* size of task in time units *)
         left : Int *         (* time units left of the job *)
         status : TaskStatus; (* for priority computation *)
color MCCTask = union Background : BackgroundTask + Rategroup : RategroupTask ;
color MCCTasklist = list MCCTask;
```

Figure 7. Colour set declarations for tasks.

Fig. 7 contains the colour set declarations for the modelling of tasks executing on the EGPC. Each task has a name (modelled by the colour set *MCCTaskName*) and can be in five different states as modelled by the enumeration colour set *TaskStatus*. There are two different kinds of tasks: *BackgroundTasks* and *RategroupTasks*. Scheduling on the EGPC is based on the concept of *major cycles* and its subdivision into *minor cycles*. Background tasks are required to be executed once within each major cycle. Rategroup tasks are required to be executed with a certain frequency (measured in minor frames) within each major cycle. The real-time requirement of the AMS is that background tasks must be completed before the end of a major cycle, and rategroup tasks must finish before the next minor cycle in which they are to be executed. Each of the two types of tasks are modelled as a record with a number of attributes. The duration of a task is captured by the *size* attribute present in both types of tasks. This attribute is used to specify the processing time (run-time) for the task when executed. This means that we do not model the detailed control-flow of tasks. Instead we model the execution of a task in an abstract way, as a period of time in which the task is in state *TaskExecuting*.

The socket places *IOQueue* and *IOStatus* in Fig. 6, represent the interface between the EGPC and the I/O processor card. Place *IOQueue* models a queue in which tasks can make requests for data to be transferred across the Mission Equipment Bus. Place *IOStatus* keeps track of the input/output status of tasks. For example, when a task *T* needs to send two messages *M1* and *M2*, a token *(T,2)* will be put in *IOStatus*, indicating that task *T* waits for two messages to be sent, and both messages *M1* and *M2* will be added to the *IOQueue*. Place *MEB* represents the interface for the mission equipment bus. The input/output processing is the aspect where the AP-3C model differs most from the generic AMS model. The modelling of input/output processing in the AP-3C model will be detailled in Sect. 5.

When it is time for tasks to start their execution, they are scheduled by the EGPC. As for the generic model, we assume that tasks request their input at the beginning, and send their output at the end of their execution. The input requests that have to be performed before the execution can start are passed by the Input/Output (I/O) processor card to the serial data bus. When all the information required has been transferred via the I/O processor, the task can be executed. When the execution is complete, the output can be performed. Then the task must wait until it has to start again.

Fig. 8 depicts page *EGPC*, subpage of the substitution transition *EGPC1* in Fig. 6. The *EGPC* has the same role as the CPU in the Mission Control Computer, namely scheduling and executing tasks. The places *Tasks* and *AllTasks* are related to the same places on page *DMS* via port-socket assignments. Place *MCCCPU* is used to model the CPU of the EPGC. The CPU may be in two states: busy executing a task, or idle. When the CPU is busy executing a task, the corresponding task will be present as a token in place *MCCCPU*. That tasks are modelled as tokens in the CPN model rather than by CP-net structure, makes the CPN model highly parameterisable. This facilitates the investigation of different scenarios. If simulation or state space analysis is to be conducted with a different set of tasks, it is only a matter of initialising the CPN model with a different set of tokens, as no change to the CP-net structure (i.e., places, transitions, arcs and inscriptions) is required.

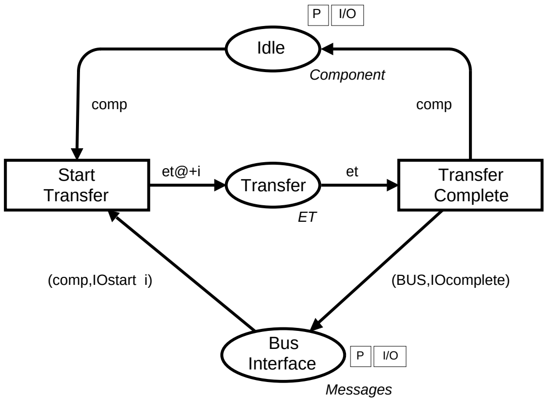
The three substitution transitions *ScheduleTask*, *InterruptTask*, and *TaskCompleted* represent the main events that may happen on the CPU. A task may be scheduled for execution, interrupted by a task with a higher priority, or may complete its execution. The associated subpages are essentially identical to the corresponding ones in the generic AMS model. Only very minor changes were made for consistency and to cater for future work. We therefore do not describe these pages here, but refer the reader to [12] for more details.

Places *Minorcycle* and *MinTime* are used to model the concept of minor cycles. Substitution transition *UpdateMajorCycle* and place *UpdatingMajor* are used to model the concept of major cycles. Page *UpdateMajorCycle* was included in the *EGPC* model, instead of staying at the *DMS* level as it only deals with changing time slots. There is a single clock corresponding to major cycles in the generic model, whereas there can be one clock per *EGPC* in the AP-3C.

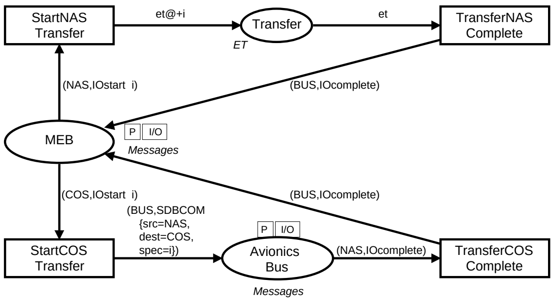
4.2. Simple Devices

The information necessary for tasks is gathered by external devices. As discussed in Sect. 2 we consider *Radar*, *Magnetic Anomaly Detection*, the *Acoustic Subsystem* and the *Communication subsystem* to be simple devices. They have the same behaviour as the *Generic Sensor* in the generic AMS model. Page *Device* modelling the simple devices is shown in Fig. 9(a). When an *IOstart* request is received via the *Bus Interface*, the device enters a *Transfer* state for the duration of the data transfer. The duration of the transfer is specified by the variable *i* which is part of the *IOstart* command. Upon completion of the data transfer, the device sends an *IOComplete* command to the *BUS* it is connected to and enter its *Idle* state.

The colour set of the port place *Idle* is *Component* (see Fig. 5). This makes it possible to parameterise the *Device* page with the specific device that it should represent. The page is instantiated via the socket place of *Idle*. For example, page *Device* is the subpage of the substitution transition *Radar* in Fig. 2. The socket place of *Idle* in this case is place *Radar* which has the initial marking *RADAR*. Thus a *RADAR* token will be present in place *Idle* for the instance of the *Device* page corresponding to the radar device.



(a) The *Device* page.



(b) The *NavigationSubsystem* page.

Figure 9. The different kinds of devices.

4.3. Navigation Subsystem

Fig. 9(b) depicts page *Navigation Subsystem*, which can receive messages from the *Mission Equipment Bus*. These messages are either for the navigation subsystem itself or for the *Communication Subsystem*. In the first case, the navigation subsystem behaves as a simple device (transitions *StartNAS Transfer* and *TransferNAS Complete*). In the second case, it passes the messages to the *Avionics Bus* (transition *StartCOS Transfer*). The *Communications Subsystem* processes the message using the *Device* page, and returns the response to the *Avionics Bus*. The response is relayed to the *Mission Equipment Bus* by transition *TransferCOS Complete*.

5. Modelling AMS Message Transmission

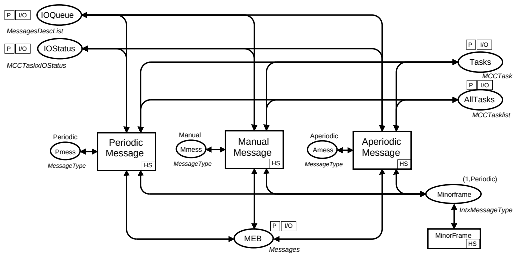


Figure 10. The *IOProcessorCard* page.

The bus considered in the generic AMS model is a basic serial data bus: it receives messages from an I/O processor, and passes them to the appropriate device. After some amount of time has elapsed, the receiving unit sends back the answer on the same data bus. The I/O processor receives and interprets it. Therefore, the model is composed of two parts (reflected in the hierarchy in Fig. 3 as two pages): the I/O processor card and the bus itself. Each task executing on the mission control computer CPU may send out messages requesting information from various subsystems. The I/O processor card sends each message over the bus and waits for the reply, have been sent. Then the messages of the next task are processed.

On the AP-3C, the I/O processor card also prepares the messages to be sent over the bus, but in a much more complex manner. The basic command/response behaviour of the bus model is the same as in the generic case. Therefore, page *MIL-STD1553* is identical to the *Serial Data Bus* used in the generic AMS model [12]. In the following, we concentrate on the modelling of the *IOProcessor* which was significantly refined when going from the generic AMS model to the AP-3C model. To understand the modelling of the I/O processor card, we explain the basic operation of the MIL-STD-1553B bus. Once again, the hierarchical net structure made things easier: a single page was replaced by a new set of related pages, without interfering with the rest of the model.

5.1. The Bus Characteristics

The dual Redundant Mission Equipment Bus (MEB) is a MIL-STD-1553B [11] digital time division multiplexed command/response serial data bus, operating at 1 Mbit per second. The master/slave protocol of the bus operates between a bus controller and other subsystems connected to the bus. It provides confirmation of delivery for data bus messages.

The AP-3C MEB uses a pre-defined framing. It contains messages that are managed by the bus controller. A major frame consists of 40 minor frames of 25 ms each that are allocated messages in a pre-determined schedule. Messages are scheduled in 3 ways: Periodic, Aperiodic and Manual. Periodic messages are transmitted every major frame. Manual periodic messages are switched on or off depending on the mode of operation. When they are on, they are transmitted every major frame. The Aperiodic messages are transmitted as required. Framing synchronization is achieved by periodic messages sent each minor and major frame. The frequency of message transmission is a function of the number of minor frames in which a message is scheduled each major frame. For example a 40 Hz message is scheduled every minor frame, a 4 Hz message is scheduled in 4 minor frames of a major one, 10 minor frames apart.

This scheduling mechanism provides reliable bus communication for critical scheduled periodic messages. Aperiodic messages require significant data transfer limited by the remaining bandwidth.

5.2. The I/O Processor Card

Page *IOProcessorCard* (see Fig. 10) depicts the DMS I/O processor card. Its role is to process the requests for input and output made by the tasks executing on the DMS CPU. This page has been considerably modified from the generic model in order to take into account the different types of messages encountered in the AP-3C: *Periodic*, *Manual* and *Aperiodic*. Among the messages ready to be sent within a minor frame, the periodic messages are sent first, then the manual messages, and finally the aperiodic ones. A sending frequency is associated with each message, determining the minor frame in which the message is sent.

Places *IOQueue* and *IOStatus* represent the interface between the tasks and the I/O processor. These two places are related via port/socket assignments to the correspondingly named places on pages *EGPC* (Fig. 8) and *TaskCompleted*. Tasks make requests for input/output by placing them in the queue modelled by place *IOQueue*. The place *MEB* represents the interface between the I/O processor card and the mission equipment bus.

The three substitution transitions *Periodic Message*, *Manual Message* and *Aperiodic Message* handle one type of message each. The corresponding sub-pages are three instances of subpage *Messages* that will be presented shortly.

The messages are sent in *minor frames*, taking some amount of time (25ms on the AP-3C). The substitution transition *MinorFrame* in page *MinorFrame* (not shown), models a switch to the next minor frame. It works as a simple periodic timer which increments the current minor frame number corresponding to the duration of minor frames.

5.3. Messages

The transmission of messages is modelled by page *Messages* shown in Fig. 11. If no message of the appropriate type remains to be sent (in place *IOQueue*), transition *NextType* can be fired, switching to the next type of message. This condition is checked in the transition guard by the SML function *NoMess*. Otherwise, the transmission of a message can start as modelled by the substitution transition *Transmission*.

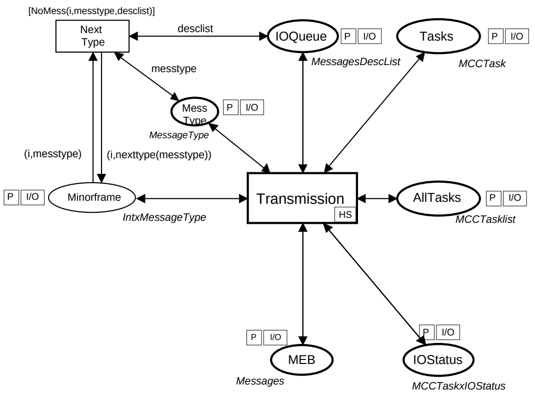


Figure 11. The *Messages* page.

Page *IOTransmit* shown in Fig. 12 is the subpage of the substitution transition *Transmission* in Fig. 11. When transition *Send* is fired, a message in the *IOQueue*, of the appropriate type and which should be sent in the current minor frame, is processed. If it is a *Periodic* message, it remains in the queue, but with the minor frame number indicating when it will be sent next. The task requesting this message enters the state *Waiting*, and the message is put on the mission equipment bus *MEB*. The acknowledgment is retrieved by transition *Receive*, which also updates the status of the task by decreasing the number of acknowledgments it is still waiting for. When a task has received all the acknowledgments it was waiting for, transition *Receive* also updates the status of the task.

Capturing the structure of messages to be sent on the *1553B* bus is an important issue. The task

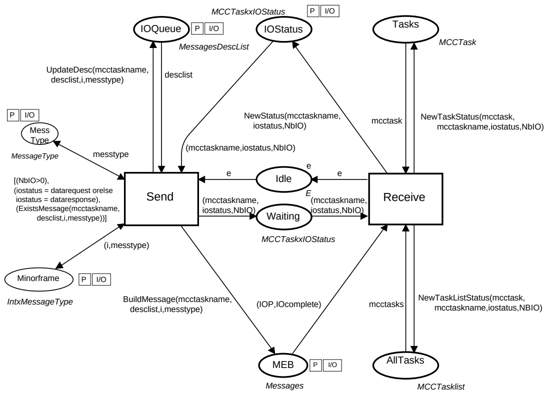


Figure 12. The *IOTransmit* page.

```
color MessagesDesc = record
    task: MCCTaskName   * msgname: MSGname *
    src: Component      * dest: Component  *
    mtype: MessageType  * nextframe: Int   *
    frequency: Int      * duration: Int;
```

Figure 13. Colour set declarations for messages.

input/output specification not only defines the requested device, and the time taken to obtain a response, as in the generic model, but must also contain more detailed information. The colour set *MessageDesc* shown in Fig. 13 is used to model the message to be transmitted across the bus.

The *task* field indicates which task is sending the message, the message is named by *msgname*, it is issued from *src* and has *dest* as its destination. The type of the message, i.e., *Periodic*, *Manual* or *Aperiodic*, is contained in *mtype*. *nextframe* describes the earliest frame in which the message can be sent, and the *frequency* allows the exact frame in which the message can be sent to be deduced: *nextframe*, *nextframe + frequency*, *nextframe + 2* $\times$ *frequency*.... The duration of messages is computed according to the following formula:

$$\begin{aligned} Duration =& 20 \times size(Message + Header) + \\ & Response\_time + Inter\_gap \end{aligned}$$

where $Response\_time = 12\mu s$ and $40 \leq Inter\_gap \leq 100\mu s$. All the times in the AP-3C CPN AMS model are expressed in μs (ms in the generic model). Such type changes are performed by updating the color declarations, writing the new SML functions operating on messages, and updating the ones of the generic model so that they operate on the more complex message structure.

6. Conclusions and Future Work

We have presented a formal specification of the mission system of the AP-3C aircraft, capturing the architecture of the system and its division into subsystems, the communication between devices and tasks executing on the main computer, and the timing related to task execution and message transfer. The formal specification is a Timed Hierarchical CP-nets. The hierarchical organisation of the model facilitated the transformation of the generic model to the AP-3C CPN model. The main differences in mission system architecture could be accommodated by changing the relationships between pages. The I/O Processor Card model could also be significantly refined without changing other parts of the model. The declarations were easily modified to incorporate the more complex structure of messages required for the MIL-STD-1553B bus. We conclude that such a modelling approach has a lot of advantages and that a well considered hierarchical design is very helpful for maintaining the model.

In the future, we plan to analyse the model with a set of tasks running on the AP-3C. In [12], we analysed a set of up to 26 tasks on the generic model. Although there are many more tasks on the AP-3C, these can be grouped into 26 *virtual nodes* which can also be considered as compound tasks. Hence, we believe that the size of the systems are very similar and that the analysis of the the AP-3C task set will be possible. We then plan to consider several *Enhanced General Purpose Controllers*, by simply creating several instances of the already existing *EGPC* page.

Acknowledgements The work presented here was done in close collaboration with DSTO and RLM. Hence, we would like to thank all the people who contributed to this specification and more particularly Raymond Kiefer from RLM systems and Rodney Dodd from DSTO.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times - A Tool for Modelling and Implementation of Embedded Systems. In *Proc. 8th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2002), Grenoble, France, Apr. 2002*, volume 2280 of *Lecture Notes in Computer Science*, pages 460–464. Springer-Verlag, 2002.

[3] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, and J. Romijn. Efficient Guiding Towards Cost-Optimality in UPPAAL. In *Proc. 7th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2001), Genova, Italy, Apr. 2001*, volume 2031 of *Lecture Notes in Computer Science*, pages 174–188. Springer-Verlag, 2001.

[4] C. S. E. Centre. www.unisa.edu.au/eie/csec.

[5] *Design/CPN Online*.

[6] A. Fehnker. Scheduling a Steel Plant with Timed Automata. In *Proceedings of 6th International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 280–286. IEEE Computer Society, 1999.

[7] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volumes 1, 2 and 3*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997. (2nd edition).

[8] L. M. Kristensen, S. Christensen, and K. Jensen. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, December 1998.

[9] J. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[10] C. D. Locke, D. R. Vogel, and J. B. Goodenough. Generic Avionics Software Specification. Technical Report CMU/SEI-90-TR-8, Software Engineering Institute, Carnegie Mellon University, December 1990.

[11] *MIL-STD-1553B Notice 3*, Jan. 1993. Digital Time Division Command/Control Multiplex Data Bus.

[12] L. Petrucci, L. Kristensen, J. Billington, and Z. Qureshi. Towards Formal Specification and Analysis of Avionics Mission Systems. In *Proceedings of Workshop on Formal Methods Applied to Defence Systems*, volume 12 of *Conferences in Research and Practice in Information Technology*, pages 95–104. Australian Computer Society, 2002.

[13] Z. Qureshi, J. Billington, and L. Kristensen. Modelling Military Airborne Mission Systems for Functional Analysis. In *Proceedings of 20th IEEE/AIAA Digital Avionics Systems Conference*, Daytona Beach, 14-18 October 2001. 12pp., CD-ROM.

[14] *RLM Systems*. www.rlmsystems.com.au.

[15] D. Science and T. Organisation. www.dsto.defence.gov.au.

[16] C. R. Spitzer. *Digital Avionics Systems – Principles and Practices*. McGraw-Hill, 1992. (2nd edition).

[17] H. Sun. Modelling and Schedulability of Real-Time Tasks with Timed Automata. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Application*, volume 4, pages 2146–2151. CSREA Press, 2001.