Formal Modelling and Analysis of Mission-Critical Software in Military Avionics Systems

Zahid H. Qureshi

Systems Engineering and Evaluation Centre
University of South Australia
Mawson Lakes Campus, Mawson Lakes 5095, South Australia

zahid.qureshi@unisa.edu.au

Abstract

A typical avionics mission system of a military aircraft is a complex real-time system consisting of a mission control computer, different kinds of sensors, navigation and communication subsystems, and various displays and stores; all interconnected by a number of serial data buses. The mission capability is increasingly implemented in the mission-critical software and the robustness of this software is vital for mission success. The complexity and real-time requirements of mission systems represent major challenges to the Australian Defence Force during new acquisitions, upgrades and maintenance. This paper describes the experiences on a joint research project between the University of South Australia and Australia's Defence Science and Technology Organisation into the modelling and analysis of avionics mission systems. The paper provides a summary of the key aspects of our previous research work on the modelling of a generic mission system using Coloured Petri Nets and the analysis of task scheduling on the mission computer. Finally, the paper briefly discusses the extension of the generic model to obtain a formal model of the mission system of the AP-3C Orion maritime surveillance aircraft.

Keywords: Avionics mission systems, formal methods, mission-critical software.

1 Introduction

The complexity of military avionics mission systems is continually increasing to meet the requirements of missions and changing operational environment. The mission capability is increasingly implemented in the mission-critical software and the robustness of this software is vital for mission success. The Australian Defence Force has experienced problems in the acquisition, upgrades and through-life support of airborne electronic mission systems, leading to cost and schedule overruns (CoA 2001). Major problems concern the integration of a large number of relatively different components and subsystems, such as radar, electronic support measures, navigation, communication and mission data processing, and that of achieving an overall optimised and operationally effective mission system.

The probable cause of the loss of the Mars Polar Lander has been traced to premature shutdown of the descent engines, resulting from a vulnerability of the software to transient signals (CIT 2000). The F/A-22 avionics have failed or shut down during numerous tests of the aircraft due to software problems. The shutdown occurs when the pilot attempts to use the radar, communication, navigation, identification, and electronic warfare systems concurrently (GAO 2003a); this has led to delays in the avionics software development and flight-testing, and to an increase in avionics development costs by over US$80 million. One of the major challenges in the F-35 Joint Strike Fighter (JSF) program includes the integration of highly advanced sensors with the avionics systems. This has contributed to the increase in the 1996 estimated cost and schedule for the JSF development phase by 56 percent and 40 per cent respectively (GAO 2003b), and an increase of an additional US$10.3 billion since the start of the system development and demonstration phase (GAO 2004).

The key issue for the Australian Defence Force is to reduce the cost of procurement and upgrades of avionics mission systems in a way that provides sufficient assurance of the system architecture and the behaviour and performance of the software-intensive mission-critical system to meet the operational requirements of the aircraft. This has motivated research into the development of a framework for analysing mission system functionality and upgrade scenarios, and validating overall system performance for future procurements and upgrades.

The aim of this paper is to describe the experiences on a joint research project between the University of South Australia and Australia's Defence Science and Technology Organisation into the modelling and analysis of avionics mission systems, conducted over a 3 year period; key aspects of this research work have been published in the public domain (Kristensen et al. 2001, Kristensen et al. 2002, Petrucci et al. 2002, Petrucci at al. 2003).

In the following section, we describe the generic mission system architecture, and in Section 3 we provide an overview of formal methods for system modelling and introduce the basic concepts of Coloured Petri Nets. In Section 4, we describe the development of a modelling framework for a generic mission system using Coloured Petri Nets, and in Section 5 we present our analysis

approach which shows how state space methods can be used in the framework of Coloured Petri Nets to reason about system properties. In Section 6, we describe how the modelling framework was used to obtain a Coloured Petri Net model for the mission system of the AP-3C Orion maritime surveillance aircraft. Finally, in Section 7 we provide conclusions on the modelling approach and discuss ideas for future work.

2 Avionics Mission Systems

The mission system for a typical combat aircraft, such as an F/A-18, is composed of many discrete avionics subsystems including radar, navigation, and mission computers. Each of these subsystems may contain further subsystems and components. Since the mission system is a very complex collection of subsystems and components, we shall employ abstraction as a technique for managing the complexity to enable the construction of models at higher levels. The generic architecture of an avionics mission system (AMS) for a combat aircraft (Locke et al. 1990) is shown in Figure 1. The AMS consists of a number of subsystems connected via a serial data bus (SDB). The control of devices, displays, and pilot controls is handled by a collection of software tasks executing on the mission control computer (MCC) which also acts as the SDB controller. The subsystems communicate by the exchange of data/messages across the SDB.

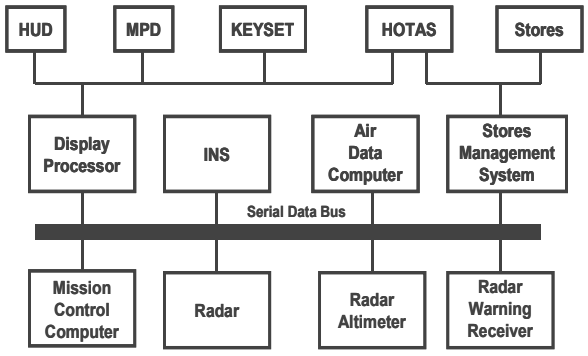


Figure 1: Generic Mission System Architecture

The controls and displays of the AMS consist of the head-up display (HUD), the multi-purpose display (MPD), the crew keyset (KEYSET), and the hands-on throttle and stick (HOTAS). These components form the human-machine interface of the AMS. The human-machine interface is controlled by the Display Process subsystem. The sensors of the AMS consist of the Air Data Computer (ADC), the Radar, the Inertial Navigation System (INS), the Radar Altimeter (RALT), and the Radar Warning Receiver (RWR). The stores contain a number of weapons such as missiles and bombs and are controlled by the Stores Management Subsystem.

Typical tasks performed by a mission control computer system may include data collection from various sensors, fusion of collected data, display of information to pilots, and controlling devices in response to inputs from the aircraft crew. One of the critical aspects of the proper functional and performance behaviour of the mission system is that the tasks must be scheduled in a way that guarantees that hard deadlines are met under all circumstances. This real-time requirement is critical to the operational performance of the mission system and hence to the success of a particular mission. Thus major concerns, when upgrading and maintaining mission systems, are the scheduling of tasks and the impact of delays associated with data transfer across the bus connecting the mission control computer and the various devices.

The key issues discussed in this paper are the scheduling of tasks on the mission control computer, and the data transfer across the data buses connecting the mission control computer and the various avionics subsystems. A typical application software task scheduling mechanism, such as for the F/A-18 and F-111 aircraft, is based on a cyclic executive (Rockwell 1992). The cyclic executive executes an application that is divided into a sequence of non pre-emptive tasks, invoking each task in a pre-determined order throughout the execution history of the application (Locke 1992). One can distinguish two types of tasks, namely, rategroup and background tasks. The rategroup tasks are periodic and have higher priority than the background tasks, which may be considered as aperiodic. The cyclic executive repeats its task list at a rate that is known as a major cycle. The major cycle is further divided into periods known as minor cycles. The major cycles have a set of tasks scheduled that must meet the required deadline in order to maintain the integrity of the mission system.

3 Formal Modelling and Analysis

3.1 Formal Methods

Formal methods are mathematically based techniques, often supported by reasoning tools, that can offer a rigorous and effective way to model, design and analyse computer systems (Bjorner and Druffel 1990). A formal method has a sound mathematical basis, typically given by a formal specification language. This basis provides the means of precisely defining notations like consistency and completeness and more relevantly, specification, implementation and correctness. It provides the means of proving that a specification is realisable, proving that a system has been implemented correctly, and proving properties of a system without necessarily running it to determine its behaviour. There are comprehensive accounts of experience on the use of formal methods in industry and research (e.g., Hinchey and Bowen 1995).

The design and validation of complex computer-based systems, for example, military avionics and space missions, should ensure the correctness of a design at the earliest stage possible. The performance of an avionics mission system is critical during flight, thereby making it a good candidate for more rigorous design and verification methods. Havelund and Lowry (2001) discuss an application of the model checker SPIN to formally analyse a software-based multithreaded plan execution module of a NASA space-craft control system. The formal verification effort had a major impact: locating errors that would probably not have been located otherwise and identifying a major design flaw.

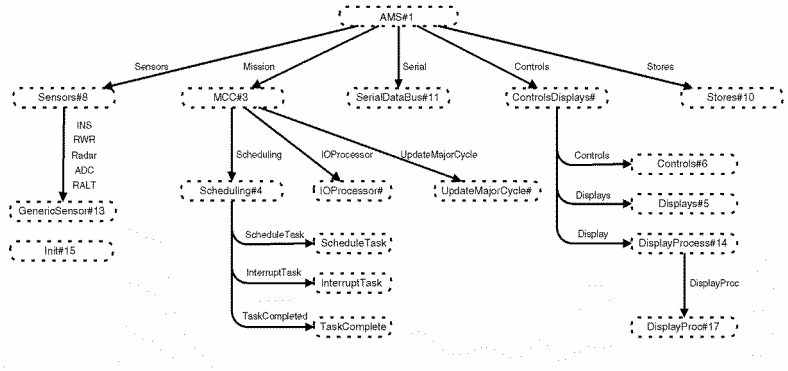


Figure 2. CPN model overview - Hierarchy page

3.2 Coloured Petri Nets

Coloured Petri Nets (Jensen 1997) are a graphically oriented modelling language for the design, specification, and verification of concurrent and distributed systems. CPNs are based on Petri Nets (Desel and Reisig 1998) and the functional programming language Standard ML (SML) (Ullman 1998). Petri Nets provide the primitives for modelling concurrency and synchronization, whereas SML provides the primitives for modelling data manipulation in systems and for creating compact and parameterisable CPN models. A CPN model of a system describes the states that the system may be in and the transitions between these states. CPN models are executable, which means that it is possible to investigate the behaviour of the system by simulations. CPN models can also be used for formal verification of systems based on state space analysis and model-checking (Jensen 1997). CPNs and the Design/CPN tool (Design/CPN Online) have been successfully applied in a wide range of application areas and many projects have been carried out in industry (Jensen 1997).

Scheduling of tasks in real time systems has traditionally been conducted using a purely algorithmic approach (Liu 2000). Recently, there has been an increasing interest in applying timed automata and model checking techniques to scheduling problems; the basic idea is to turn the scheduling problem into a reachability problem that can be solved by analysis tools using a state space search (Petrucci et al. 2002). The advantages 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 properties, such as functional correctness. Hence, it represents an integrated approach to the analysis of the system.

4 CPN Model of a Generic Mission System

In this section, we describe our modelling framework and approach by providing an overview of the CPN model of the generic mission system and give some representative examples of modelling at the different levels of abstraction in the CPN model; the reader is referred to Kristensen et al. (2001) for details.

4.1 Modelling Framework Overview

A CPN model can be structured into a number of hierarchically related modules (in CPN terminology called pages) with well-defined interfaces between them. The *hierarchy page* giving the overall structure of the CPN AMS model is depicted in Figure 2. Each node in Fig. 2 represents a page (module) of the CPN AMS model, and is named according to the page in the CPN model that it represents. An arc leading from one node to another node means that the latter is a subpage (submodule) of the former. The page *AMS* is the top-most page in the CPN model.

The CPN model consists of five main parts which correspond to the five immediate subpages of the AMS page: the *MCC* page and its subpages models the Mission Control Computer, the *Sensors* page and its subpages models the sensors, the *ControlsDisplays* page and its subpages models the man-machine interface, the *Stores* page models the stores and the Stores Management System, and the *SerialDataBus* page models the Serial Data Bus.

Figure 3 depicts the *AMS* page and provides the most abstract view of the model. This page consists of five *substitution transitions* (drawn as rectangles with an HS tag in the lower right corner) and two *places* (drawn as ellipses). The substitution transitions *Mission Control Computer*, *Sensors*, *ControlsandDisplays*, *StoresManagementSystem*, and *SerialDataBus* correspond to the five main parts of the AMS system. Each of the

sesubstitution transitions (and its surrounding places) is related to a *subpage*. The subpage of a substitution transition provides a more detailed description of the compound activity/component represented by a substitution transition. Place *SerialDataBus* is used to model the data transfer across the serial data bus. Place *CDS* represents the interface between the controls and the storage management system. Each of the places in Fig. 3 is so-called *socket places* used to link the subpage of the substitution transition and the *AMS* page. Socket places are assigned (linked) to *port places* on the subpage of the substitution transition.

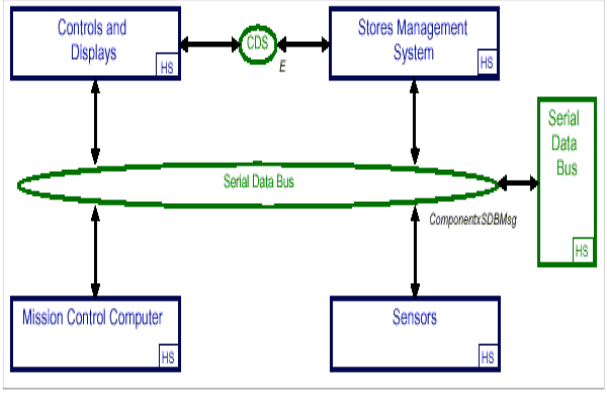


Figure 3: The AMS page

An addressing scheme has been developed to capture the AMS architecture, identity the components of the AMS, and to model the interaction between the components. Components can be hardware devices (e.g., sensors) as well as software processes/tasks (e.g., the tasks executed by the MCC). In addition to making it possible to identify components, the virtue of this addressing scheme is that new components can easily be added to the CPN model without having to make global modifications to it. This means that new tasks on the MCC as well as new hardware devices can be incorporated into the CPN model with only local modifications. A sample of the colour set definitions used to realise the addressing scheme are shown in Figure 4. The colour set definitions are written in the Standard ML programming language and are similar to type definition found in programming languages. The colour set *MCCTaskName* identifies the different tasks executing on the mission control computer. The colour set *BackgroundTask* contains the attributes:

```
color MCCTaskName = with
  AircraftFlightData | Steering
| RadarControl       | TargetDesignation
| TargetTracking     | WeaponSelection
| WeaponTrajectory   | WeaponRelease
| HUDDisplay         | MPDHUDDisplay
| MPDTacticalDisplay | MPDStoresDisplay
| MPDStatusDisplay   | KEYSET_HOTAS
| RWRControl         | RWRThreatControl
| BuiltInTest ;

color BackgroundTask =
      record name : MCCTaskName *
             size : Int *
             left : Int ;

color RategroupTask =
      record name : MCCTaskName *
             rate : Int *
             next : Int *
             size : Int *
             left : Int ;

color MCCTask = union
   Background : BackgroundTask +
   Rategroup : RategroupTask ;
```

Figure 4: Colour set definition for tasks

- *name* - the name of the task;
- *size* - total size of the task when executed (measured in time units);
- *left* - how much of the task remains to be executed

In the colour set *RategroupTask*, the *rate* attribute specifies the frequency of a rategroup task (as the number of minor cycles between execution of the task), and the *next* attribute keeps track of the next minor cycle in which the task should be executed.

4.2 Mission Control Computer

Figure 5 depicts the MCC page which is the most abstract part modelling the mission control computer. The MCC is the subpage associated with the substitution transition Mission Control Computer from Fig. 3. The page has two substitution transitions: *Scheduling* represents the scheduling mechanism on the mission control computer and *IOProcessor* represents the IO processor of the mission control computer.

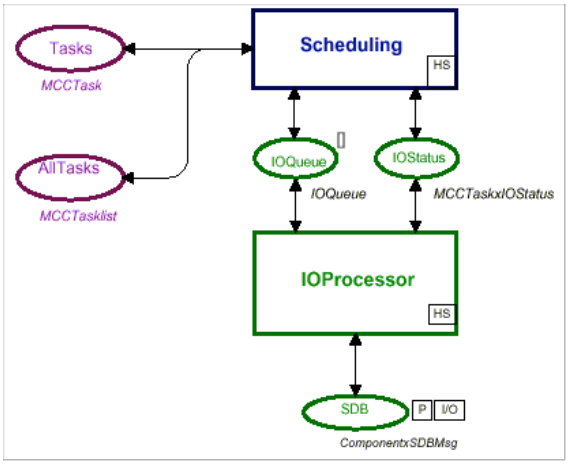


Figure 5: Mission Control Computer (MCC) page

A marking (state) of a CP-net is represented by a distribution of tokens on the places of the CPN model. The kind of tokens that can reside on a place is determined by the colour set of the place. The colour set of a place is by convention written below the place. The tokens initially present on a place are specified by the initial marking of the place. The initial marking of a place is by convention written above the place and omitted if the place is initially empty (i.e., contains no tokens).

The input socket places *Tasks* and *AllTasks* of the substitution transition *Scheduling* are used to represent the task of the mission control computer. Place *Tasks* has the colour set *MCCTask*, and each task on the mission

control computer is represented as a token of colour set *MCCTask* on the place *Tasks*.

The place *AllTasks* contains a list of all the tasks on the mission control computer. This list is used to access all tasks on the mission control computer and determine which task will be executed next. The *Tasks* place ensures that if there is a choice of the next task to execute on the mission control computer, the CPN model represents all possible such choices.

The socket places, *IOQueue* and *IOStatus*, represent the interface between the mission control computer and the IO processor. The place *IOQueue* is used to model a queue in which tasks can make requests for data to be transferred across the serial data bus. The place *IOStatus* keeps track of the input/output status of tasks, and to signal (using an interrupt) that data requested by a given task is now available. The place *SDB* represents the interface between the serial data bus and the IO processor.

4.3 Task Scheduling and Execution

The *Scheduling* page, shown in Figure 6, models the general scheduling mechanism on the mission control computer (MCC). This page is the subpage of the substitution transition *Scheduling* from Fig. 6. It consists of seven places and three substitution transitions. The port places *IOStatus*, IOQueue, *Tasks* and *AllTasks* are assigned to the identically named socket places on page *MCC*.

The place *MCCCPU* is used to model the state of the processor (CPU) in the mission control computer. Figure 7 shows the colour set declarations used to model the state of the CPU. The state of the CPU is modelled by the colour set *CPUState*. The CPU may either be *Idle* or *Busy*, i.e., executing a task. The colour set *BusyState* is a product where the first component is used to specify the task that the CPU is busy executing. The second component is used to record the time at which the task started executing. This information is used to compute how much of the task was completed in case the current task is interrupted by a higher priority task. The colour set *CPUState* is timed. This means that tokens of this colour set will carry *time stamps*. These time stamps will be used to specify the time at which the task has run to completion. The initial marking of place *MCCCPU* is a token with colour *Idle* corresponding to the CPU initially being idle.

The place, *Minorcycle*, is used to keep track of the current minor cycle. As indicated by the initial marking of this place, the system starts in minor cycle 1. The place *MinTime* represents the minimum amount of time a task should spend on the CPU before it can be pre-empted by a task with a higher priority.

The substitution transition *InterruptTask* models how a task executing on the CPU can be interrupted by a task with a higher priority, and the substitution transition *TaskCompleted* describes the completion of a task. The pages associated with these substitution transitions are described in detail in Petrucci et al. (2002).

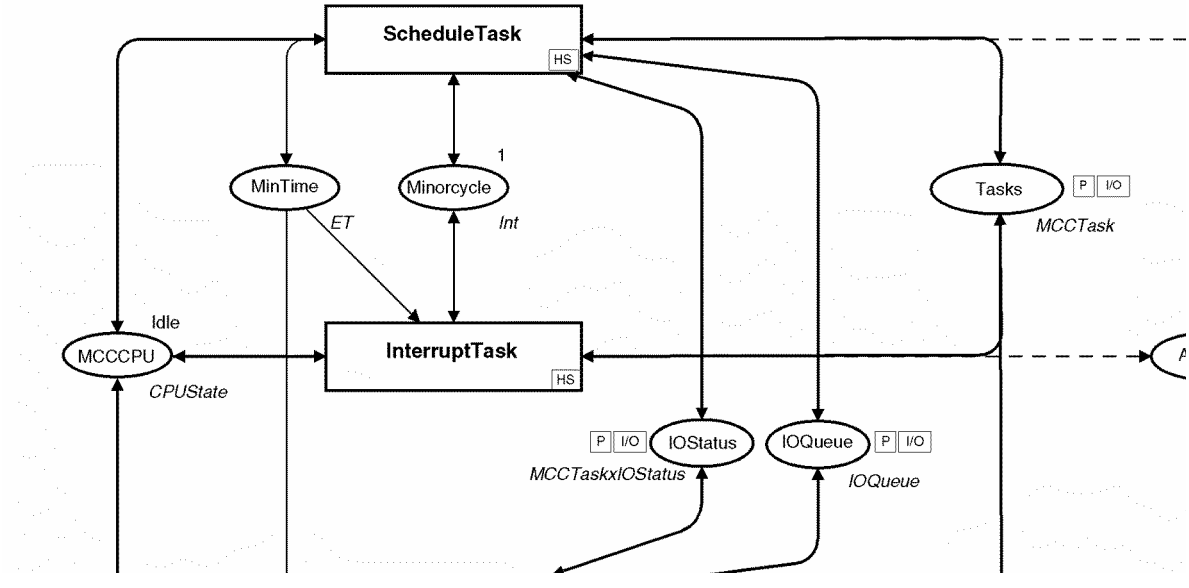
```
color CPUState = union
       Idle +
       Busy : BusyState timed;
```

Figure 7: Colour set definitions for CPU state

4.4 Serial Data Bus

The *Serial Data Bus* page shown in Figure 8 comprises three places and two transitions. There are two places *Idle* and *Busy* describe the state of the *Serial Data Bus*. The third place, *SDB*, represents the information transiting on the *Serial Data Bus*.

The *Serial Data Bus* is initially in the *Idle* state. When a request arrives on the *SDB* place from the I/O processor, it starts transmitting the request to the appropriate device (transition *Start Transmit* occurs). The serial data bus will then change its state form idle to busy by placing a token



in the place *Busy*. When the request has been transmitted, the *Serial Data Bus* becomes Idle again and signals that the request has been completed so that the I/O processor handles the next request (transition *Transmit Complete* fires). The places *Idle* and *Busy* ensure that a transfer on the data bus can only start if no other transfer is already being processed.

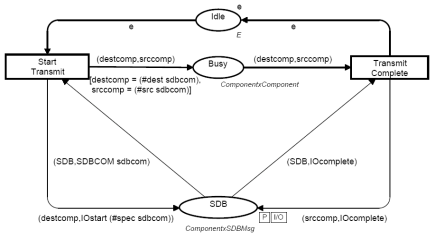


Figure 8: Serial Data Bus page

4.5 Sensors

The information necessary for tasks is gathered by various sensors such as radars. The execution of tasks causes data to be transferred between sensors, control, displays, and stores. The interaction between the *MCC* and the sensors consists of data transfers. Since we model the mission system at a high level of abstraction, the various sensors can be modelled as shown in the *GenericSensor* page (Figure 9).

When a request arrives on the *Serial Data Bus*, the relevant *Sensor* is activated and the data transfer starts (transition *Start Transfer* occurs). Each sensor takes a certain amount of time to operate and process information, and thus must remain in the *Transfer* state during this time. This is achieved by the time stamp given to the token created in place *Transfer* when transition

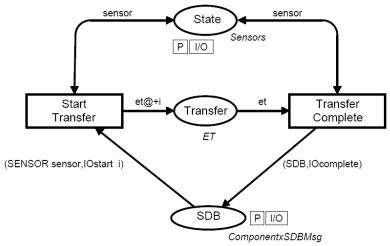


Figure 9: GenericSensor page

5 Analysis of the CPN Model

The analysis of the CPN AMS model is based on the state space method of Coloured Petri Nets as supported by the Design/CPN tool. The primary focus of the analysis has been to determine whether the tasks to be scheduled on the mission control computer are completed in time, and if this is the case, provide a schedule for the set of tasks. In addition to this, the size of the input/output queues of the I/O processor has been considered. The basic idea behind state space methods is to construct a directed graph (called the state space) with a node for each reachable state of the CPN model and an arc for each transition between states. Since the state space contains all reachable states it represents all possible executions of the CPN model. In this section, we provide a summary of our analysis approach and results which have been reported in detail in Petrucci et al. (2002).

The problem of finding a schedule can be formulated as finding a path in the state space leading from the initial state to a state where the major cycle has ended and all

Task Set	Tasks	RG	BG
S1	Displays and Controls HUDDisplay, MPDHUDDisplay, MPDTacticalDisplay, MPDButtonResponse, ChangeDisplayMode, MPFStoresDisplay, MPDStatusDisplay, KeysetResponse, HOTASDesignation, HotasBombButton	6	4
S2	S1 + Built-In Test PeriodicBIT, BITFailureWarning, InitiatedBIT	7	6
S3	S2 + Radar Control RadarSearch, radarTracking, RadarInitiateTracking	9	7
S4	S3 + Targeting DesignateTarget,ConfirmDesignation, TargetTracking, TargetSweetening	10	10
S5	S2 + Threat Response PollRWR, ThreatResponseDisplay	8	7
S6	S5 + RWR Control RWRProgramInput, RWRProgramming	9	8
S7	S6 + Weapon Control (except WeaponRelease) InputWeaponSelection, WeaponSelectionProc, AutoCCIPtoggle, WeaponTrajectory, ReinitiateTrajectory	10	12
S8	S7 + Targeting Designatetarget, ConfirmDesignation, TargetTracking, TargetSweetening	11	15

Table 1: Set of Tasks used for Analysis

Start Transition occurs. The token in place *Transfer* cannot be consumed before this amount of time has elapsed. When the sensor has terminated, i.e., transmitted

tasks were completed in time. To make state space analysis feasible, we started out by selecting a small set of tasks and gradually introduced additional tasks. Also, we experimented with different priority policies for tasks

accessing the CPU and for input/output. Table 1 lists the different sets of tasks taken from Locke et al. (1990) that are used for the analysis. The RG column gives the number of rategroup (periodic) tasks in a given set. The BG column specifies the number of background tasks in the set.

Table 2 gives the size of the state space for the different sets of tasks listed in Table 1. The Nodes column gives the number of nodes in the state space, and the Arcs column gives the number of arcs in the state space. The IOSS column gives the maximum number of requests in the I/O requests queue at the I/O processor observed in the state space. The IOSP column gives the maximum number of requests in the I/O queue along a path in the state space corresponding to a schedule for the tasks. The considered pre-emption and queuing policy allowed requests from rategroup tasks to overtake requests from background tasks in the IO queue, and both rategroup and background tasks had assigned priorities.

Set	Nodes	Arcs	IOSS	IOSP
S1	77,982	127,316	6	4
S2	78,734	128,715	7	6
S3	485,054	811,734	9	7
S5	144,780	235,769	10	10
S6	142,022	234,257	8	7
S7	409,888	702,831	9	8

Table 2: Standard State Space Generation

Various state space analysis techniques were considered, for example, the use of depth-first state space generation allowed the S4 and S8 task sets to be analysed, as this led to significantly fewer states to be considered (Petrucci et al. 2002).

Our analysis focuses on task scheduling of the mission control computer and only considers a single major cycle. This is sufficient because all tasks are required to be executed at the end of a major cycle. Analysis results showed that, for example, the rategroup task *WeaponRelease* cannot execute in time i.e. it fails to meet its deadline. We have demonstrated how analysis of scheduling and input/output queue of an avionics mission system can be done using state spaces.

6 AP-3C Orion Mission System Modelling

The AP-3C aircraft mission system upgrade provides enhanced mission capabilities and extends the P-3C life-of-type to 2015 (DMO 2002). The AP-3C aircraft is operated by the Maritime Patrol Group of the Royal Australian Air Force, and its mission roles include anti-subsurface and anti-surface warfare, surveillance, search and rescue, and maritime strike.

A high-level block diagram of the AP-3C avionics mission system architecture is shown in Figure 10 (RAAF 1996). It consists of a Mission Equipment Bus (MEB) that provides inter-communication channels between the following sub-systems: Navigation, Acoustics, Magnetic Anomaly Detection (MAD) and Radar. The Data Management System (DMS) provides specialised interfaces to the following sub-systems: Armament/Ordinance (ARM/ORD), Electronic Support Measures (ESM), and Infrared Detection System (IRDS). The operators are provided with a high-resolution display and entry panel directly from the DMS. The Communication (Comm) sub-system interfaces with these sub-systems via the Avionics Equipment Bus (AEB), which is connected to the MEB via the Navigation sub-system. The MEB and the AEB are dual redundant serial data busses based on the MIL-STD-1553B (DOD 1993). The DMS of the AP-3C plays the same role as the mission control computer in the generic mission system.

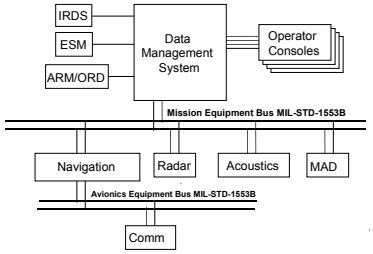


Figure 10: AP-3C AMS Architecture

The DMS is a centralised mission control and management sub-system. It is a complex multiprocessor system consisting of several Enhanced General Purpose Controllers (EGPC), custom computing devices and supporting software. Two input/output (I/O) processor cards provide the interface between the EGPCs and the MEB. The DMS provides the overall AMS management and normally acts as the MEB bus controller (RAAF, 1996).

The DMS software consists of a set of software components, which process sensor data and input from controls, performs necessary mission-oriented computations and provides outputs to the displays and other avionics equipment (RAAF 1999). Typical software components include the executive, navigation, stores management, display control, and data management. The runtime executive schedules and dispatches the execution of control tasks and services interrupts during various operations. The runtime executive component is typically responsible for the following functions: application software (task) scheduling, interrupt management, input/output scheduling and error management. The DMS executive software is based on a commercial Ada run-time kernel (RAAF 1999). The scheduling policy is pre-emptive and is executed by priority in a round-robin fashion (Rational 1995).

An EGPC sends a number of messages to the I/O processor for transmission to the addressed remote terminal (RT) (RAAF 1996), e.g. Navigation sub-system or Radar subsystem, via the MEB. The MEB minor frame rate (described in the next section) sets a real-time clock interrupt to the I/O software. At the beginning of each new minor frame, an interrupt occurs, and the I/O processor starts issuing the messages for that frame.

The AP-3C CPN model has been constructed based on the framework represented by the CPN AMS model briefly described in Section 4. From the two architectures

depicted in Figures 1 (generic) and 10, it follows that although the components in the two systems are basically the same, the structure is quite different. There are two buses 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. In this section, we provide an overview of the model only, and the reader is referred to Petrucci et al. (2003) for details.

Finally, we need to consider two sorts of subsystems (or *device*) in the AP-3C mission system, rather than one. Because of the level of abstraction chosen for the model, we refer to subsystems or devices that have just a single connection to a bus as a *simple device*, such as, 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.

The hierarchy page of the AP-3C CPN model is depicted in Figure 11; a parallel should be drawn between this page and the generic AMS model hierarchy in Figure 2. The four main differences between the CPN models of the generic AMS and the AP-3C, as reflected in the hierarchy pages, are as follows:

- The *Sensors* and *Generic Sensor* pages from the generic AMS model have been combined and then split into two: the simple devices (in the Device page) 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, input/output processing was modeled by page *IOProcessor*. In the AP-3C model, input/output processing 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, and the 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 bus. As usual we model

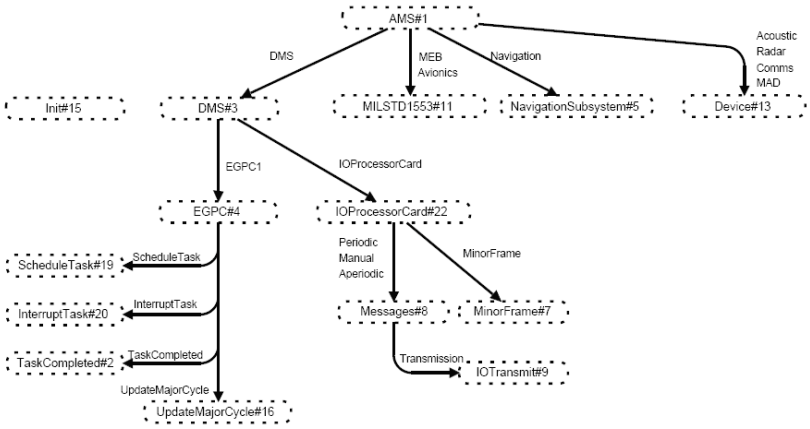


Figure 11: Hierarchy page of the AP-3C model

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

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 be 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.

The top-level AMS page is shown in Figure 12 and corresponds to the most abstract level in the AP-3C CPN model. This page provides a high-level architectural view of the AP-3C AMS similar to the informal block diagram in Figure 12. All the transitions (rectangles) are substitution transitions, indicated by the HS (hierarchical substitution) tag 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 busses. 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.

Figure 13 depicts the DMS page which is the most abstract part modelling the data management system. The page 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 modelling

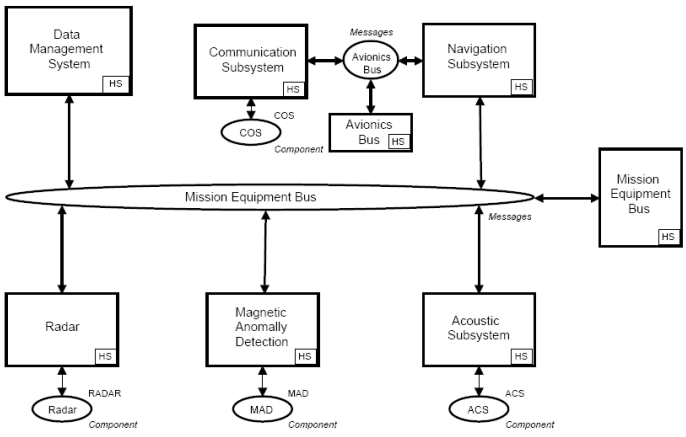


Figure 12: AMS page of the AP-3C model

details of processing and messages transmission on the IO Processor Card is described in detail in Petrucci et al, (2003). The AP-3C aircraft uses up to 4 EGPCs. Our model can easily cater for this by including the required number of EGPC substitution transitions (e.g. *EGPC1* to *EGPC4*). Thus, several instances of the *EGPC* page may be used concurrently.

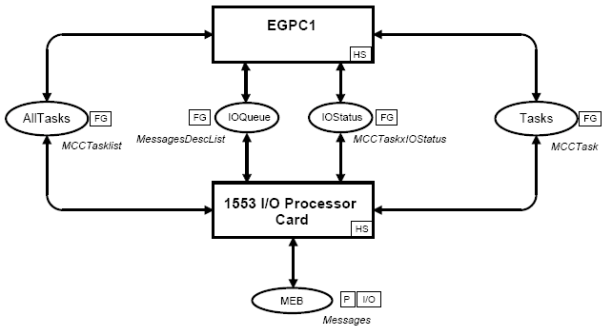


Figure 13: DMS page of the AP-3C model

7 Conclusions and Way Ahead

This paper has provided an overview of the experiences on a joint research project between the University of South Australia and Australia's Defence Science and Technology Organisation into the modelling and analysis of avionics mission systems, conducted during the period 2000-03.

We have described the development of a formal model of a generic mission system using Coloured Petri Nets. The main outcome is the capture of system domain knowledge, understanding of the mission system architecture and the interrelationships between the various avionics subsystems. We have demonstrated how analysis of task scheduling of an avionics mission system can be done using state spaces. A virtue of our modelling approach is that the CPN model is highly parametric, which makes it easy to analyse different sets of tasks and in this way investigate the impact of adding tasks to the mission control computer. Another advantage of our modelling approach is that specific scheduling mechanisms can easily be changed and an analysis of their impact can be conducted.

We have shown how the hierarchical constructs of CP-nets and Design/CPN that were successfully used to model the generic airborne mission system could be readily applied to model the AP-3C mission system. The initial CPN model of AP-3C provides a basis to perform analysis of the mission system architecture and is focused on the Data Management System task scheduling and data transfer across the Mission Equipment Bus. The CPN model can serve as an unambiguous executable specification of the DMS. Since the CPN models are executable, the behaviour of the DMS can be observed by simulating the CPN DMS model. This can prove to be an efficient way of gaining and maintaining knowledge on the operation of the DMS. The CPN DMS model can be used to analyse what-if scenarios before the actual system integration. These what-if scenarios could be related to functional as well as performance aspects of the DMS.

A major problem is the state-space explosion, for example the state space of the CPN AMS model grows with the increase in the number of tasks. In order to perform more efficient state space analysis, the use of advanced methods such as the sweep-line methods (Christensen et al. 2001) should be investigated.

Safety-critical software for aviation is typically DO-178B (RTCA 1992) level A or B (BAE, 2004). Mission-critical software applications must also be developed using the guidelines of DO-178B. However, unlike safety-critical applications, mission-critical software is typically DO-178B level C or D (BAE, 2004). Formal methods can play an important role in the design evaluation of mission-critical software and systems.

Furthermore, there is a need to investigate other methods and techniques for the modelling and analysis of avionics mission systems, such as: Avionics Architecture Description Language (SAE-AADL) and associated MetaH tool (Vestal 1997), model based verification and lightweight formal methods (Gluch and Weinstock 1998), and formal verification techniques and tools, for example, SPIN model checker (Holtzmann 1997), PVS theorem prover (Crow et al. 1995), and another promising verification tool (under development) - the Hierarchical Verification Environment (HiVE) (Cant et al. 2005),

Finally, in addition to the research on modelling that has been presented in this paper, we recommend a number of critical research areas to complement the overall program for research on avionics mission systems, namely, advanced avionics architectures including safety, availability, fault-tolerance and growth issues, avionics data bus architectures and performance issues, real-time schedulability and timing analysis, and software and hardware design assurance.

8 Acknowledgements

This research was conducted under the Long Range Research Task scheme, Task No. LRR 00/061, during the author's association, as the task manger, with the Australian Defence Science and Technology Organisation. The author would like to acknowledge Professor Billington, University of South Australia on the research collaboration and guidance with the application of Coloured Petri Nets. The author would also like to acknowledge Lars Kristensen, Aarhaus University for the research on the development of the generic avionics mission system CPN modelling framework and Laure Petrucci, Laboratoire Spécification et Vérification, ENS de Cachan, for the formal analysis and refinement of the framework to the AP-3C aircraft, during their sabbatical at the Computer Systems Engineering Centre, University of South Australia. Finally, acknowledgements are due to Raymond Kiefer and Scott Simmonds from Tenix/RLM and Adacel Technologies respectively, for the technical discussions and design information on the AP-3C Data Management System, and to Flight Lieutenant Jon Postle, AP-3C DMS Manager, Royal Australian Air Force.

9 References

BAE (2004): *Safety-critical vs. mission-critical: Understanding the difference*, CompactPCI and AdvancedTCA Systems, January, OpenSystems Publishing, http://www.compactpci-systems.com/articles/id/?263; Accessed 19 July 2006.

Bjorner, D. and Druffel, L. (1990): Position statement: ICSE-12 Workshop on Industrial Experience Using Formal Methods, *Proceedings of the 12th International Conference on Software Engineering*, 264-266, March 26-30, Nice, France.

Cant, T., Mahony, B., McCarthy, J., and Vu. L. (2005). Hierarchical Verification Environment, *Tenth Australian Workshop on Safety Critical Systems and Software*, SCS 2005, Cant, T. (Ed.), Conferences in Research and Practice in Information Technology, 55: 47-57, Australian Computer Society.

Christensen, S., Kristensen, L.M. and Mailund, T. (2001): A Sweep-Line Method for State Space Exploration, Proc. TASCAS'01, LNCS, 2031:450-464, Springer Verlag.

CIT (2000): Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions, Jet Propulsion Laboratory, California Institute of Technology.

CoA (2001): Major Capital Equipment Project Delays or Cost Overruns, Additional Budget Estimates – Defence Portfolio, February, Defence White Paper Projects, Senate Foreign Affairs, Defence and Trade Legislation Committee, Canberra, Commonwealth of Australia.

Crow, J., Owre, S., Rushby, J., Shankar, N. and Srivas. M. (1995): A Tutorial Introduction to PVS, *Proc. of the Workshop on Industrial-Strength Formal Specification Techniques (WIFT'95)*, USA, Computer Science Laboratory, SRI International.

Desel, J. and Reisig, W. (1998): Place/Transition Petri Nets, Lectures on Petri Nets I: Basic Models, *Lecture Notes in Computer Science*, 1491:122-173, Springer-Verlag.

Design/CPN Online: Computer Tool for Coloured Petri Nets, University of Aarhus, http://www.daimi.au.dk/designCPN/. Accessed 7 Mar 2000.

DMO (2002): Projects, Air 5276 Phase 2A - P3 Update Implementation, Defence Materiel Organisation, http://www.defence.gov.au/dmo/asd/air5276/air5276p2.cfm, Accessed 25 July 2002.

DOD (1993): MIL-STD-1553B Notice 3, 31 Jan 1993, Military Standard, Digital Time Division Command/Response Multiplex Data Bus, Department of Defense, Washington DC.

GAO (2003a): DOD Should Reconsider Decision to Increase F/A-22 Production Rates While Development Riske Continue, Report to Congressional Committees, March, GAO-03-431, General Accounting Office, Washington, D.C.

GAO (2003b): Defense Acquisitions: Assessments of Major Weapon Programs, Report to Congressional Committees, May, GAO-03-476, General Accounting Office, Washington, D.C.

GAO (2004): Status of the F/A-22 and Joint Strike Fighter Programs, Testimony before the Subcommittee on Tactical Air and Land Forces, Committee on Armed Services, House of Representatives, March 25, GAO-04-597T, General Accounting Office, Washington, D.C.

Gluch, D.P. and Weinstock, C.B. (1998): Model-Based Verification: A technology for Dependable System Upgrade, Technical Report CMU/SEI-98-TR-009, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Havelund, K. and Lowry, M. (2001): Formal Analysis of a Space-Craft using SPIN, *IEEE Transactions on Software Engineering*, 27(8):749-765.

Hinchey, M,G. and Bowen, J. P. (Eds.), (1995*): Applications of Formal Methods*, International Series in Computer Science, UK, Prentice Hall.

Holtzmann, G. (1997): The Model Checker SPIN, *IEEE Transactions on Software Engineering*, 23(5):279-295.

Jensen, K. (1997): *Coloured Petri Nets. Basic Concepts, Analysis Method and Practical Use* (*Vol. 1-3)*, Monographs in Theoretical Computer Science, Second Edition, Springer-Verlag.

Kristensen, L. M., Billington, J. and Qureshi, Z. H. (2001): Modeling Military Airborne Mission Systems for Functional Analysis, *Proc. 20th IEEE/AIAA Digital Avionics Systems Conference*, Daytona Beach, Florida, 14-18 October.

Kristensen, L. M., Billington, J., Petrucci, L., Qureshi, Z. H. and Kiefer, R. (2002): Formal Specification and Analysis of Airborne Mission Systems*, Proc. 21st IEEE/AIAA Digital Avionics Systems Conference*, Irvine, California, 27-31 October.

Liu, J. (2000): *Real-Time Systems*, New Jersey, Prentice-Hall.

Locke, C.D., Vogel, D.R. and. Goodenough, J. B. (1990): Generic Avionics Software Specification, Technical Report CMU/SEI-90-TR-8, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Locke, C.D. (1992): Software Architecture for Hard Real-Time Applications: Cyclic Executive vs. Fixed Priority Executives, *The Journal of Real-Time Systems*, 4:37-53.

Petrucci, L., Kristensen, L. M., Billington, J. and Qureshi, Z. H. (2002): Towards Formal Specification and Analysis of Avionics Mission Systems, In Lakos, C., Esser, R., Kristense, L. M., and Billington, J. (Eds.), *Proc. Workshop on Formal Methods Applied to Defence Systems*, June, Conferences in Research and Practice in Information Technology, 12:95-104, Australian Computer Society.

Petrucci, L., Billington, J., Kristensen, L. M. and Qureshi, Z. H. (2003): Developing a Formal Specification for the Mission System of a Maritime Surveillance Aircraft, *Proc. 3rd International Conference on Application of Concurrency to System Design*, Guimaraes, Portugal, 18-20 June.

RAAF (1996): Mission Equipment Bus (MEB) Protocol Specification. Report F6250.00.151, 11 March 1996, (CDRL-ENG-46-ACS-ICD-0074, Rev D 10 Jan 2000). Project Air-5276 Royal Australian Air Force, Department of Defence, Australia.

RAAF (1999): Software Design Document for the Operating Systems CSCI, Report 7371902, Rev B, 26 February 1999, (SDRL-ENG-65D-03-03, CDRL-ENG-32-DMS-SDD-0423, 2 July 1999). Project Air-5276 Royal Australian Air Force, Department of Defence, Australia.

Rational (1995): Runtime Systems Guide VADScross M68000, ver: 6.2.3.0, June, Rational Software Corporation, Santa Clara, California.

Rockwell (1992): F/RF-111C Avionics Update Program, 1992, Software Design Document for the Mission Computer Operational Flight Program, Volume I, Rockwell International Corporation.

RTCA (1992): *Software Considerations in Airborne Systems and Equipment Certification*, DO-178B, Washington DC, RTCA.

SAE-AADL: SAE AADL Information Site, A Society of Automotive Engineering Standard, http://www.aadl.info/. Accessed 11 April 2006.

Ullman, J. (1998): *Elements of ML Programming*, New Jersey, Prentice-Hall.

Vestal, S. (1997): MetaH Support for Real-Time Multi-Processor Avionics, Joint Workshop on Parallel and Distributed Real-Time Systems (WPDRTS/OORTS '97), IEEE Computer Society.