18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

Modelling Airborne Mission Systems using the Architecture Analysis and Design Language

Sioutis, C [1], **T. Nguyen** [1]

[1] *Airborne Mission Systems, Air Operations Division, Defence Science and Technology Organisation, South Australia*
Email: christos.sioutis@dsto.defence.gov.au

Abstract: One of the critical success factors for integration of a Defence system is a good architectural design. A right architecture can help ensure that a system will satisfy its key operational requirements as well as its quality attributes such as real-time performance, reliability, security, and maintainability. A bad architecture on the other hand is a recipe for disaster (Clements et. al. 2001). Emerging properties of systems such as scheduling, fault tolerance, and security can cause significant problems in integration of complex Defence systems. Life-cycles are becoming evolutionary as components of systems are upgraded to avoid obsolescence, but such upgrades must be done in the context of system impact (Allen et. al. 2002).

Software intensive acquisition projects are historically considered the most risk prone in the Defence domain and often incur schedule delays, cost overruns, and reduced operational capability (Commonwealth of Australia 2004). Capability systems have a “life cycle” that begins with the identification of the need to address a current or prospective capability gap. This need is progressively translated into a working capability system that is operated and supported until it is ultimately withdrawn from service. Architecture modelling and analysis can potentially be incorporated to the Defence Capability Life Cycle (DCLC) (Commonwealth of Australia 2006) for assessing technical risks associated with proposed system architectures.

This paper describes initial research conducted for modelling and analysing Airborne Mission Systems (AMS). The Architecture Analysis and Design Language (AADL) was chosen for this purpose because it is specifically geared for model-based development and analysis of real-time embedded systems. The AADL is a textual and graphical language that models the architecture of systems as an assembly of software components mapped on an execution platform. The primary advantage for using the AADL is that it provides different ways of expressing a model being developed. Graphical AADL is useful for development and visualization, text AADL is useful editing and parameterisation, XML AADL is useful to conduct various types of analysis with different third party tools. This allows finding problems early and potentially saving costs throughout the system's entire life cycle. This paper introduces the AADL and then describes the two case studies employed to understand and apply it.

The aim of the first case study was to gain an appreciation of how to develop an AADL model in general. It was decided to develop an AADL model that was related to the air domain but not be overly complex. This reasoning led to the modelling a model-helicopter in flight, which includes a pilot controlling the helicopter via a controller. The second study focused on investigating how to specifically model a subset of the mission system hardware and software of a Royal Australian Navy S-70B-2 Seahawk helicopter.

This research revealed that an AADL model must be developed based on what is the specific question being investigated. Models intended to simply describe a system can afford to be more abstract in order to highlight higher level architectural elements. Conversely, models intended for specific analysis must have an adequate fidelity and be populated with any parameters needed to perform that analysis.

Keywords: *Architecture Analysis and Design Language, Modelling, Analysis, Airborne Mission Systems*

1. INTRODUCTION

As the complexity of AMS increases, their integration becomes more challenging. There are currently processes in place during the acquisition of such systems designed to assess their Technical Readiness Level (TRL) and System Readiness Level (SRL) and identify risks to the program (Smith et. al. 2004). However TRLs and SRLs are insufficient as tools for identification of system integration risk because they do not provide an explicit risk framework (Nandagopal 2006). Architectural modelling and analysis on the other hand can serve as potential tools for identifying technical risks early. Performance modelling at the architectural level provides a significant means of reducing the cost of development through early discovery of issues. When combined with an approach that generates system integration code compliant with the models, rapid evolution is achievable (Allen et. al. 2002).

This paper describes the initial steps of a long range research program aimed at understanding how to apply architectural level modelling and analysis to assess the SRL and TRL of AMS. Some of the questions specifically being investigated are: What information is required from the supplier in order to generate adequate architectural models? What additional information is necessary to populate such models sufficiently in order to perform analysis? How can one model the architecture of an AMS in a meaningful way? What types of analysis can be performed to a particular architectural model? How do results from such analysis reflect on the SRL and TRL of the system?

Section 2 of this paper provides a brief introduction to the Architecture Analysis and Design Language (AADL) which is used for the modelling. Section 3 describes the first case study conducted which looks at modelling a model-helicopter in flight. Section 4 describes the model for a subset of the mission system of a Seahawk helicopter. Finally, Section 5 draws conclusions from this work, as well as insights into future directions.

2. THE ARCHITECTURE ANALYSIS AND DESIGN LANGUAGE

The Architecture Analysis & Design Language (AADL) was developed by the international Society for Automotive Engineers (SAE). It is specifically geared for modelling and analysis of real-time embedded systems. Using AADL it is possible to define interfaces, connections and aggregation of system components (software, hardware and their associated mapping). When the standard AADL constructs are not enough, it can also be extended using a custom annex. The AADL standard provides a textual and graphical language used for system modelling; an extensible Markup Language (XML) and XML Metadata Interchange (XMI) format; translation profiles for computer programming languages; and a UML 2.0 profile (Embedded Computing Systems Committee 2004).

An AADL *specification* is comprised of a series of *declarations* that describe a particular aspect of a system. The AADL also allows the use of *packages* to aid in organizing declarations into related sets. Hence, one is able to place all declarations relating to a particular section of the system into a single package. The AADL requires viewing a system as a range of distinct interconnected components. A *component type* is used to specify the externally viewable interface of a component. It contains features like communication *ports* and *properties*. A *component implementation* defines exactly how it works inside. Implementations typically encapsulate a number of *subcomponents* and *connections* leading to a full implementation hierarchy. Additionally, component implementation can specify different *modes* of operation, with mode-specific property values and internal configurations.

Software components are used to model the architecture of the software employed within a system. The *data* component models a data type used within a system, it can hold information as well as *subprograms* with application logic. A *thread* models a flow of control that executes sequential instructions. *Thread groups* can be used to group threads logically if required. Finally, a *process* models a virtual address space in memory that defines a partition whose boundaries are enforced at run-time.

Execution platform components model the mechanisms used to execute the software components. They can represent physical as well as virtual entities of a real system. The *processor* component is responsible for scheduling and executing software threads. A *memory* component stores binary images of any subprograms, data, and processes available to a processor. A *bus* models a communication pathway between different components. It supports configuration parameters such as throughput, error rate, and quality of service. Finally, a *device* component allows a system to interface with the external environment. A device also has a special ability, in that it can be connected directly with both hardware and software components.

The AADL uses *systems* to group components together. They allow breaking up a large system into smaller interconnected *subsystems*. A system provides an encapsulation barrier where any internal components are not able to be directly connected to any external components without going through its pre-defined *ports*. These model conduits of control (events) and information flow (data) in and out of a system, their declaration provides the system's interface signature. Therefore, systems can be interchangeable as long as they have a compatible external interface. Similarly to devices, system components can be connected to both software and hardware components simultaneously, hence providing the vehicle for bridging the software/hardware gap. The AADL also allows the flexibility of having some components described in greater detail while others are left more abstract.

The AADL supports a number of different types of analysis. Semantic checks are used to scan through the AADL specification code and detect common errors. Architecture analysis tools consider the entire system architecture and generate statistics about different aspects of the model. However, they require the model to be populated with various properties derived from measurements conducted on the real system. For example, there are tools to check that the system architecture caters for incomplete sensor readings, resource boundaries, safety requirements and security requirements. Additionally, the data flow latency tool compares the latencies of measured data flow implementations to the maximum latency required by the corresponding end-to-end data flow declaration. A sufficiently defined AADL system component can be *instantiated*. This causes all parameters in the model to be initialized, all sub-systems instantiated as well, and all inter-connections formed and cross-checked. Errors are generated when a system model is incorrect or incomplete.

In the context of scheduling analysis, an application is considered to consist of a set of processors, shared resources, and tasks. Using the task information, two types of scheduling analysis can be performed, scheduling simulation and feasibility tests. Scheduling simulation involves predicting the task to which the processor should be allocated for each unit of time. This is then checked to ensure that the tasks meet their processing deadlines. Feasibility tests are performed when a scheduling simulation takes a long time to compute. They use periodic tasks to measure the processor utilization of the system. This analysis requires populating the model with data about task deadlines, task execution times, processor speeds, scheduling policy, timing requirements and task synchronization details.

3. CASE STUDY: T-REX MODEL HELICOPTER

Modelling a T-REX 450XL (Align 2005) (from now referred as T-REX) model helicopter was chosen for the first investigation in applying the AADL model to the air domain. This work aimed to assess the AADL's expressive power, usability, learning curve, documentation, and availability of online support methods.

The T-REX is marketed as a 3D model helicopter because its power-to-weight ratio allows it to perform rather interesting acrobatics (e.g. it can fly upside down) that a normal helicopter cannot normally do. The T-REX comprises of many small parts. However, these can be combined into a small number of major components that are either electrical, electronic, or mechanical. They are: *Battery*, supplies approximately 12V power and generally lasts around 30 minutes of flight time; *Motor*, drives both the main and tail rotors; *Regulator*, regulates power from the 12V battery into stable 5V for other components aboard; *Antenna*, used to pick-up the signal emitted by the operator's controller; *Radio Frequency (RF) Receiver*, converts the RF control signal into electrical signals of other on-board components; *Electronic Speed Controller*, converts the battery power into three-phase power used to drive the motor; *Servos*, mechanical devices that change their position based on an electrical control signal; *Gyroscope*, measures changes in inertia caused by movement and generates a signal to stabilize; *Rotor blades*, provide the lift for flying (main rotor) and turning (rear rotor); *Swashplate*, translates control signals into changes of the main rotor blade pitch;

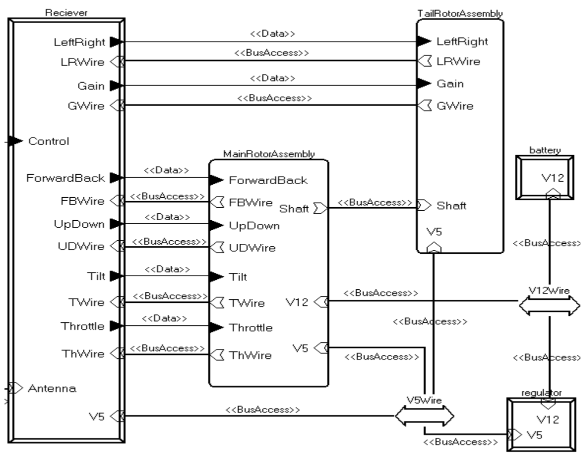

Figure 1. T-REX High Level Graphical AADL

The helicopter has a number of internal components and it became necessary to push them down one level into two major subsystems, the *MainRotorAssembly* and the *TailRotorAssembly*. Figure 1 illustrates the top most level of the model where a device called *Receiver* receives the external control signal and splits it up into six separate channels each on its own *Wire* bus. Each channel is then directed into the appropriate subsystem. The power distribution is also modelled using separate wire buses.

Developing the T-REX AADL model proved a relatively straightforward exercise. It involved obtaining information about the composition and operation of the T-REX and then translating this into a model. The model was developed in OSATE (Carnegie Mellon University 2008) which is an integrated development environment based on the open source Eclipse framework (Eclipse Foundation 2008) with additional functionality geared specifically for AADL modelling and analysis. Only a fraction of the concepts available in the AADL were necessary to provide sufficient means to model the T-REX at a suitably detailed level. The OSATE tool was easy to understand and operate and provided instant feedback in the form of warnings/errors during development. The OSATE documentation included information about how to use the tool itself and an electronic copy of virtually the entire AADL standard. The graphical AADL editor in OSATE is incomplete but still under active development. This means that even through the graphical editor was used, a substantial amount of AADL code also needed to be written by hand. The T-REX model initially contained no software components because the helicopter itself is a simple electromechanical device controlled by an external radio frequency signal. Subsequent additions to the model investigated how to turn the T-REX into a mini UAV. Hence new components were integrated to the model for this purpose. Examples include a microcontroller with autopilot software, an on-board GPS, a three-axis gyroscope, and a safety override switch.

4. CASE STUDY: S-70B-2 SEAHAWK HELICOPTER

An opportunity was subsequently identified to attempt to apply the AADL to model a part of a currently operational mission system, specifically that of the Seahawk helicopter. This is due to readily available access to documentation and source code of the Seahawk’s Display Graphics Unit (DGU).

Figure 2. S-70B-2 Seahawk helicopter (Navy 2009)

The RAN operates S-70B-2 Seahawk helicopters (shown in Figure 2) designed to meet their requirement for destroyer and utility helicopters. Specific roles include antisubmarine warfare, anti surface surveillance, targeting mission requirements, search and rescue and transport of materials and troops. The Seahawk can operate in any maritime region of the world in support of the parent ship's deployment schedule. A typical Seahawk mission involves low level operations over the sea irrespective of time or weather conditions, often recovering to a wet ship's deck that pitches and rolls in heavy seas.

The Seahawk's mission system uses two DGU modules for graphics generation on cockpit monitors. The two DGU modules are set up for hot-redundancy. When the primary DGU called the Bus Controller (BC) fails, then the secondary DGU called the Remote Terminal (RT) takes over. The DGU functions and capabilities include: Data bus control for MIL-STD-1553B; input/output interfacing; state synchronization and hot-redundancy swapping between the two DGUs; collection and control of avionics system status/health; data processing, collection and forwarding for navigation, guidance, and communication; composite video output with internally generated symbology; and track coordination and maintenance for self, parent ship and sensors.

The mission system software of the Seahawk comprises over 200,000 lines of augmented Ada83 source code. Many different Ada tasks are concurrently running, each responsible for taking care of a specific aspect of the mission computer's processing. The spread of tasks being executed also depends on whether the DGU is running in BC or RT mode (Dodd 2006). The AADL model developed for this research describes a subset of the hardware and software architecture of the two DGUs and their connection through the 1553 bus. The model has been split up into a number of packages defined in separate files, each AADL package focuses on a specific aspect of the system. This made it much easier to locate, update and cross-reference the various components during development.

The hardware package integrates hardware components which are connected via two main internal buses (system and display) as shown in Figure 3. An external MIL-STD-1553 bus is used for synchronization between the two DGUs. Due to the sheer number of messages sent between the BC and RT DGUs via the 1553 bus, a separate package was defined specifically for listing the relevant data. The type and direction of data is then encapsulated within an AADL port group component. The memory package models a shared memory chip within the DGU where all processors store their data and communicate through. The common memory is the only medium of communication between components residing on different internal DGU buses. Each DGU uses three identical AAMP processors. These are modelled in the processors package and they required to be connected to a DGU bus. The 1553 I/O card used in the DGU is also included in the processor package because it performs its own processing for successfully managing the connection to the 1553 bus. It is modelled as a device component that requires access to both the DGU and 1553 buses.

The *Tasks* package defines the software components used for modelling the tasks being executed in the DGU. The task implementations are then defined and linked with the appropriate subprogram component. Some tasks in the DGU are executed periodically (every few milliseconds). Such tasks are modelled using a thread with a periodic dispatch protocol property and a period attribute to indicate how often it is executed. Other tasks are triggered through stimuli from an operator or other components. These are modelled using a thread with an aperiodic dispatch protocol property and a trigger event port.

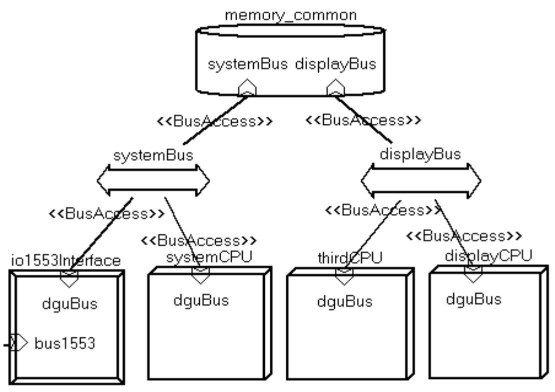

Figure 3. DGU hardware graphical AADL

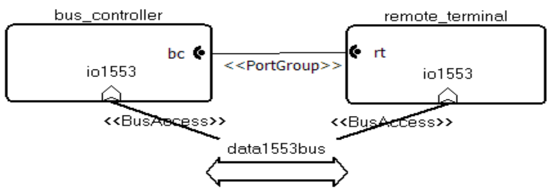

Figure 4. Partial mission system graphical AADL

The software package encloses all the software into a system. Since all tasks running in the DGU share a single common memory, they have been modelled as running within the same process address space. Two modes of operation are defined, the *bus_controller* and the *remote_terminal*. These correspond to the DGU operating in BC and RT mode respectively. This is important because there is a different task spread for each mode. Some tasks execute only in the BC mode, others only in the RT mode, and others in both. *Aperiodic* tasks need to be triggered through an event and three types of events have been identified: First, events originating from tasks executed by the DGU main task scheduler; Second, events originating from the 1553 bus; and third, events originating from actions performed by the operator. All components defined in the different packages are integrated together in the *Display_Graphics_Unit* system. This effectively links together the hardware and software systems as subcomponents in a larger system. The final model incorporates the two such systems connected via the 1553 bus and a port group connection as shown in Figure 4.

Modelling the Seahawk DGU brought forward different issues to the ones encountered with the T-REX. In this case there was a large amount of ad-hoc information available with no particular structure. For example, the available source code could not be built and executed with standard Ada compilers because it used custom language additions. This meant that it was necessary to manually browse the Ada code and try to ascertain what tasks there were and how they operated. It therefore proved difficult to extract the required information in order to construct an accurate model and populate it with parameters that depict the actual operation. In order to obtain this information it would be necessary to either devote additional resources to this problem, or to try to obtain more detailed information from the manufacturer.

Earlier research by Dodd (2006) utilised coloured petri-nets to model the DGU scheduler. Dodd was able to conduct analysis on his petri-net model and obtain simulated processor utilisation values that were comparable to the real system. Dodd's advantage in this case was that he focused specifically on the scheduler and was able to tailor his model such that it generated the information that he was looking for. Conversely, in the research reported here an attempt was made to use the AADL to model the entire system architecture, which includes the DGU software and hardware. Conceptually this seems reasonable. However, practically the complexity required to model every aspect of the DGU in AADL is very large. This complexity can be reduced by abstracting parts of the system out, but as the fidelity of the model decreases its utility for analysis diminishes.

5. CONCLUSIONS AND FUTURE DIRECTIONS

This paper describes the initial steps of a long range research program aimed on understanding how to apply architectural level modelling and analysis of AMS. The primary advantage for using the AADL is that it provides different ways of expressing a model being developed. Graphical AADL is useful for development and visualization, text AADL is useful for editing and parameterisation, XML AADL is useful to conduct various types of analysis with different third party tools. The AADL was found to be rich, easy to understand and use. However, its practical application becomes increasingly difficult as size and complexity increases. The model developer needs to have extensive knowledge and understanding about the system and its operation.

This research has therefore revealed that an AADL model must be developed based on the specific question being investigated. One can therefore focus on a specific aspect of a system under investigation and create a model specifically for that purpose that is tailored to generate the information required for analysis. Models intended to simply describe a system can afford to be more abstract in order to highlight higher level architectural elements. Conversely, models intended for specific analysis must have an adequate fidelity and be populated with any parameters needed to perform that analysis.

The mission system software of new platforms currently in acquisition by Defence (like the upcoming Airborne Early Warning and Control (AEW&C) *Wedgetail*) are being developed using the Service Oriented Architecture (SOA) development methodology (Foster et. al. 2007). Future research will investigate if the AADL can be utilised to model and analyse such systems. It is believed that the layered architecture evident in SOA-based systems will enable applying the AADL at a specific implementation layer, effectively bypassing the complexity of lower layer implementations. Successful completion of this work will allow DSTO to provide better technical advice to Defence in regards to the operation and possible future upgrades of the Wedgetail and/or other SOA-based systems in future acquisition or service.

REFERENCES

Align (2005), *T-REX 450XL CCPM instruction manual*, Taichung, Taiwan.

Allen R., S. Vestal, D. Cornhill, and B. Lewis (2002), *Using an architecture description language for quantitative analysis of real-time systems*, In Proceedings of the 3rd International Workshop on Software and Performance, pages 203-210, Italy.

Carnegie Mellon University (2008), *Open Source AADL Tool Environment (OSATE)*, URL:http://la.sei.cmu.edu/aadl/currentsite/tool/osate-down.html

Clements, P., R. Kazman, and M. Klein (2001), *Evaluating Software Architectures*, Addison Wesley Professional, US.

Commonwealth of Australia (2004), *Defence Electronic Systems Sector Strategic Plan*, Department of Defence, Australia.

Commonwealth of Australia (2006), *Defence Capability Development Manual*, Department of Defence, Australia.

Dodd R. (2006), Coloured Petri Net Modelling of Task Scheduling in Airborne Mission Systems, Technical Report, DSTO-TR-1897, Defence Science and Technology Organisation, Australia.

Eclipse Foundation (2008), *Eclipse*, URL: http://www.eclipse.org

Embedded Computing Systems Committee (2004), *The Architecture Analysis & Design Language (AADL) Standard*, International Society for Automotive Engineers, US.

Foster K., A. Iannos, G. Lawrie, P. Temple and B. Tobin (2007), Exploring a Net Centric Architecture using the Net Warrior Airborne Early Warning and Control Node, Technical Report, DSTO-TR-2093, Defence Science and Technology Organisation, Australia.

Nandagopal N. (2006), *Systems integration challenges for Defence*. Defence Magazine, October, pages. 26-27, Australia.

Navy (2009), *S-70B-2 Seahawk*, Department of Defence, On-line accessed 30/01/09 URL: http://www.navy.gov.au/Seahawk

Smith J., G. Egglestone, P. Farr, T. Moon, D. Saunders, P. Shoubridge, K. Thalassoudis, and T. Wallace (2004), *Technical risk assessment for Australian Defence projects*, Technical Report, DSTO-TR-1656, Defence Science and Technology Organisation, Australia.