Modeling and Rapid Prototyping applied to the Upgrade of Mission-Critical Applications using AADL

Gustavo Rodríguez, Alfons Crespo, Jérôme Hugues, and Manuel Amor

***Abstract*—Upgrade mission-critical applications such as the Operational Flight Program (OFP) of avionics embedded computer systems is a complex task. Requires software engineering techniques particularly difficult from capturing requirements until its implementation. Modeling and Rapid Prototyping with Architecture Analysis & Design Language (AADL) are techniques that can help upgrade the OFP mainly at the stage of formulating and validating requirements.**

***Index Terms*—Avionics Systems, Operational Flight Program, Modeling, Rapid Prototyping, AADL.**

I. Introduction

UPGRADE mission-critical applications such as the OFP running in the aircraft's Mission Computer (MC) is a complex task. These are generally long life applications, thus, in the best case, it is upgrading applications based on software engineering techniques particularly difficult. It is not uncommon for an OFP change or block update to take as long as one year before released to the field. These software engineering techniques consist of a series of stages beginning with requirements capture to implementation and testing [1].

In general, upgrading an OFP may be due mainly to two reasons:

- correction of errors detected during system operation and
- proposed improvements of the system by adding new functionality.

In either of the two cases it is necessary to evaluate how the upgrade will affect the existing avionics system.

Developers must apply software refinement in order to proceed from a high-level abstract model to a final executable software system by adding more details over time. Modeling and Rapid Prototyping are techniques that contribute to the design and incorporation of these upgrades [2] [3].

In software engineering, there are two main approaches to prototyping. One approach is to develop a draft implementation (a throwaway or rapid prototype) in an attempt to learn more about the requirements on the software, throw the prototype away, and then develop production quality code based on the experiences from the prototyping effort. The other approach (evolutionary prototyping) is to develop a high quality system from the start and then evolve the prototype over time. Unfortunately, there are problems with both approaches.

G. Rodríguez is with Real-Time Systems Group, Department of Telecommunications, Rio Cuarto National University. Rio Cuarto, Argentine. e-mail: grodriguez@ing.unrc.edu.ar

A. Crespo is with Industrial Informatics and Real-Time Systems Group, Department of Systems Data Processing and Computers, Polytechnic University of Valencia. Valencia, Spain.

J. Hugues is with Department of Mathematics, Computer Science, and Control of the Institute for Space and Aeronautics Engineering (ISAE). Toulouse, France.

M. Amor is with Real-Time Systems Group, Department of Telecommunications, Rio Cuarto National University. Rio Cuarto, Argentine.

Model Driven Engineering (MDE), a software development methodology promoted by the Object Management Group (OMG), focuses on creating and exploiting domain models simplifying the process of design and promoting communication between customers and developers.

The AADL [4] together with Modeling and Analysis of Real-time and Embedded systems (MARTE) [5] are considered as valuable candidates to support an MDE method for the design and the implementation of avionics systems.

The AADL was released by the SAE in November 2004, and is an architecture description language dedicated to embedded real time systems in safe-critical domains. It provides an industry-standard, text and graphical notation with precise semantics to allow early function and quality properties verification at the model level and automatic code generation from the models, which greatly improves productivity and assures high reliability. The core language is enhanced by annex languages that enrich the architecture description. One of them is directly connected to avionics systems. ARINC653 annex: define patterns for modeling avionics systems.

The ARINC-653 standard [6] introduces the concept of partitioning. It consists in isolating applications in space and time so each partition has an address space and a time slice to execute their code. Using this partitioning architecture, several applications, at different criticality or security levels, can be collocated on the same processor [7] [8] [9] .

This paper describes the application of modeling and rapid prototyping based on AADL in upgrading mission-critical applications such as an OFP, applied mainly at the stage of formulating and validating requirements.

An overview of AADL on how it is used in upgrading an OFP is presented in this paper. Section II presents some background, as well as the goals of the upgrade an OFP. Next, in Section III, we discuss a rapid prototyping methodology applied to an avionics system and related work. In Section IV there is a presentation of our case study using prototyping and model refinement with AADL and Ocarina. Finally, Section V ends this paper with a conclusion.

II. Background

A. Modeling and Rapid Prototyping

Modeling is a way of thinking about problems using models organized around real-world ideas [10].

Models are abstractions that portray the essentials of a complex problem or structure by filtering out nonessential details, thus making the problem easier to understand.

Abstraction is a fundamental human capability that permits us to deal with complexity.

A prototype is any form of specification or implementation of hardware or software (or both) that is built or designed for evaluation purposes. When building a prototype, a designer has typically in mind to design multiple copies once the design is sufficiently evaluated. All prototypes have in common that they are executable. Prototypes are also useful for formulating and validating requirements, resolving technical design issues, and supporting computer aided design of both software and hardware components of future designs.

Rapid prototyping refers to the capability to implement a prototype with significantly less effort than it takes to produce an implementation for operational use. It is a way to collect data and feedback for changing requirements, unveil deviations from users constraints early, trace the evolution of the requirements, improve the communication and integration of the users and the developers, provide early feedback of mismatches between proposed software architectures and the conceptual structure of requirements.

Prototypes can either be developed as temporarily designs that are not being used after some evaluations have provided the designer with valuable insights, or they can directly evolve into a product version of the respective design. Each of these approaches has its advantages and disadvantages. It will depend on the designers constraints which type is the best for a certain project.

Software prototyping has many variants. However, all the methods are in some way based on two major types of prototyping: Throwaway Prototyping and Evolutionary Prototyping.

- Throwaway: prototypes are built to validate a concept, prior to implementing the real system. The throwaway approach is used to refine requirements.
- Evolutionary: prototypes tend to become the final product. Prototypes are refined to create more accurate ones. The last prototype actually corresponds to the final system. Then, feed-back on the system may be provided at various levels and the model is the main reference for describing the system.

The most common problem with throwaway prototyping is managerial, many projects start developing a throwaway prototype that is later, in a futile attempt to save time, evolved and delivered as a production system. This misuse of a throwaway prototype inevitably leads to unstructured and difficult to maintain systems.

Notable examples of work in prototyping in embedded systems development include PSDL [11] [12], Rapide [13] [14] and AADL and Ocarina [15] [16]. PSDL is based on having a reusable library of Ada modules which can be used to animate the prototype. Nevertheless, it seems that this approach would preclude execution until a fairly detailed specification was developed. Rapide is a useful prototyping system, but it does not have as much flexibility to integrate as easily with other tools as we desired. AADL and Ocarina are the methodology used for this work.

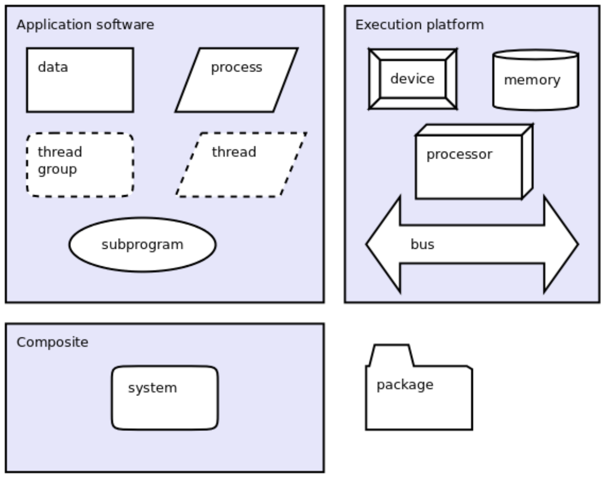

Fig. 1: AADL components to develop an AADL Model

B. AADL

The AADL is a standard architecture modeling language, developed by and for the avionics, aerospace, automotive, and robotics communities. It uses component-based notation for the specification of task and communication architectures of real-time, embedded, fault-tolerant, secure, safety-critical, software-intensive systems.

The language and its associated tools are used to model, analyze, and generate embedded real-time systems.

The AADL offers *threads* as schedulable units of concurrent execution, *processes* to represent virtual address spaces whose boundaries are enforced at runtime, and *systems* to support hierarchical organization of threads and processes. The AADL supports modeling of the execution platform in terms of *processors* that schedule and execute threads; *memory* that stores code and data; *devices* such as sensors, actuators, and cameras that interface with the environment; and *buses* that interconnect processors, memory, and devices. The Figure 1 shows the different components to develop an AADL model.

Threads can execute at given time intervals (periodic), triggered by events (aperiodic) and paced to limit the execution rate (sporadic), by remote subprogram calls (server), or as background tasks. These thread characteristics are defined as part of the thread declaration.

Application components interact with other application components and devices exclusively through defined interfaces. A component interface consists of ports for unidirectional flow of data (data ports for unqueued state data, and event data ports for queued message data) and events (event ports) between threads and to and from devices; synchronous subprogram calls between threads, possibly located on different processors; and access to data that is concurrency

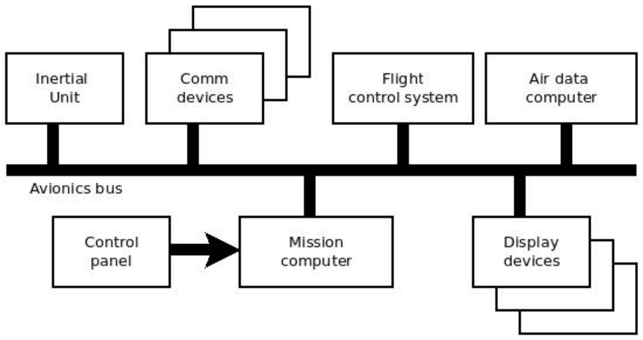

Fig. 2: Generic avionics system architecture

controlled. Data port connections can be specified to perform mid-frame (immediate) communication within the same dispatch period, or phase delayed (delayed) communication for data to be available after the deadline of the originating thread. These semantics ensure deterministic transfer of data streams between periodic threads, an important feature for embedded control systems. Deterministic data transfer means that a thread always receives data with the same time delay; if the receiving thread is over- or under-sampling the data stream, it always does so at a constant rate.

C. Avionics Systems

An avionics system is a collection of subsystems that support flight through several functions to ensure that the aircraft's mission to be carried out effectively. Figure 2 shows a generic avionics system architecture.

The total system may be considered to comprise a number of major subsystems, each of which interacts to provide the overall system function. Major subsystems themselves may be divided into minor subsystems or equipment which in turn need to operate and interact to support the overall system. Each of these minor subsystems is supported by components or modules whose correct functional operation supports the overall system. The overall effect may be likened to a pyramid where the total system depends upon all the lower tiers.

An avionics system includes communications, navigation, the display and management of multiple systems, and the hundreds of systems that are fitted to aircraft to perform individual functions. An OFP is the software program embedded in the MC which enables to avionics system to perform its interactive tasks as designed.

The period of development of a complete cycle of an OFP (that is, the upgrade from one version to another) will depend on many factors such as, among others, the complexity, the operational need and the availability of human and material resources. However, average intervals can be set between 1 and 1.5 years.

Moreover, a new OFP's version provides improvements in the operation of the systems, or incorporating new capabilities, and, of course, all this will result in a medium-to long-term, operational and logistical implications to be derived from its implementation.

The complete cycle management of a system that requires operating software includes all activities and tasks ranging from the conception of the need for a new program until it is fully implemented and operational. All these activities can be divided into three phases clearly identified and separated from each other, that determine the so-called software life cycle on a cascade-type model, this implies that is not possible to tackle the tasks corresponding to a phase if you have not completed the entire phase above.

This lifecycle definition remains cascaded along the different tasks that compose each one of the phases, but with greater flexibility, allowing varying degrees of overlap between tasks based on the type of development and workflow in which lays each project or requirement. The three phases of the cycle are:

1) Phase I: Definition
2) Phase II: Development
3) Phase III: Implementation

The objective of **Phase I Definition** is to study, analyze and evaluate the implications that may involve incorporating the proposed changes. This stage is critical because it will determine the other two and includes those activities and tasks leading to obtain the widest possible knowledge for each anomalies or proposals for change and its possible solutions to be adopted for each case, so that, once satisfied the objective of this phase is to address the **Phase II Development**.

During **Phase I Definition** is essential to use tools to transform software engineering anomalies and proposed changes into a particular requirement to be incorporated into the new version. The rapid prototyping helps to understand and get a broader knowledge of the anomalies and proposals for change in order to turn into a formal requirement.

The motivation of rapid prototyping techniques in the **Phase I Definition** may be:

- reduce cost and development time and
- increases security and reliability of an avionics system.

III. A Rapid Prototyping Methodology for Avionics Systems

A typical OFP development process (upgrade) consists of developers (software engineers, programmers) and customers (pilots, analysts, etc.) collaborating together to develop several options to evaluate for each requirement to be incorporated into the new version. These options are often coded in high level languages or graphical tools based on PC, with purposes of having throwaway prototypes. The customer then evaluates and discusses on the different options and a requirements list is made.

This list becomes the basis for the software requirements specification that the development team uses while develops and tests the software. At the end of this process, which usually takes more than 1 or 1.5 years, the customer can see the resulting code in the flight simulator and later in flight. If problems are found at that time or during flight tests, an extensive period of time is often required to make changes and corrections.

A. Eliciting Requirements and Building Prototypes

Specification-based prototyping allows an iterative approach to building a formal specification. Initial versions of the specification can focus on the desired control behavior and ignore complicating details, such as sensor or actuator failures.

The specification can be evaluated in an equally simplified environment containing failure free sensors and actuators. As the understanding of the system and its environment deepens, both the specification and the environmental models will be refined. The following paragraphs outline the activities that occur as the models are constructed and refined [17].

As mentioned in previous sections, these throwaway prototypes are made in high-level languages, usually the development is ad hoc without considering any details of the implementation.

As a result, even at this early stage of system specification, an executable prototype is available. In this fashion, the analysts and stakeholders can actively evaluate and test the high-level requirements of the system early in the development cycle.

B. Refining the Model and Aspects of Human-Machine Interface

Initially all requirements are considered together and an impact analysis of the system is done. Importantly, there may be requirements that are not entirely related to the philosophy of the system. At this time is when they are discarded.

The process includes the grouping of requirements in sets by affinities or complexity. With the set of requirements defined, the prototype is refined increasingly incorporating more and more realism, such as sensors and actuators. In general, it is actually a prototype for each set of requirements and as a result of all this has several throwaway prototypes that provide a total behavior of the system change proposal.

In cases where there are different options for a given requirement should be considered conducting various prototypes to select one of the options.

Finally, the requirements that contain human-machine interfaces have special treatment. The prototype should not only functionally characterize the requirement but must additionally allow a vision of it. Prototypes are generally more configurable and with the rise of graphical visualization tools based prototypes traditional languages are increasingly obsolete.

IV. Case Study: Upgrading an OFP using Rapid Prototyping with AADL

Upgrade an OFP can be a labor, sometimes even more complicated than creating a new one from scratch. The Model Driven Engineering (MDE) is a comprehensive approach to software system development (upgrade) that supports the entire product lifecycle. The AADL is considered in a number of projects aiming at defining and implementing tools support for MDE.

In this section we detail a methodology for upgrading an OFP using rapid prototyping with AADL. Our case study is updating an OFP, incorporating a set of navigation requirements to the original design.

Each requirement is analyzed individually and are considered different options to incorporate. The analysis includes the development of several prototypes to simplify the decision which option is best suited to the original design.

In our case we take the navigation requirements, specifically related to the takeoff and landing of aircraft and routes. It was decided to make several throwaway prototypes to emulate those conditions.

The process continues with AADL modeling of each of the options on the requirements.

A. AADL Model Creation

The process implements the following (possibly iterative) methodology to define and refine each requirement or set of requirements:

- data types and related functions to operate on them
- supporting runtime entities (`threads`) and interactions between them (through `connection` and `ports`)
- association of functions to threads
- mapping of threads onto `processes` and `processors` to form the deployed system.

The AADL allows graphical and textual modeling. At first, some graphics can be made to permit to be a starting point for developers thereafter textual models are more effective.

The Software Engineering Institute at Carnegie Mellon University developed the extensible Open Source AADL Tool Environment (OSATE). OSATE consists of a set of plug-ins for the open source Eclipse Platform [18].

As a result of this process an AADL model is obtained for each prototype, including functional requirements to incorporate another as well as some non-functional properties.

B. AADL Model Validation

Validation of a prototype by an AADL model takes on several forms. For this case study we used two forms:

1) semantic analysis validating the consistency of an AADL model,
2) analysis an AADL model with respect to functional and nonfunctional properties of the embedded system represented by the model.

Semantic analysis is performed using AADL compiler Ocarina. Ocarina checks that the given AADL model is conforming to the AADL grammar and that some additional restrictions are verified:

- All event or data ports are connected,
- All threads are either periodic or sporadic,
- All shared data use a concurrent access protocol.

With respect to nonfunctional requirements was performed a schedulability analysis using Cheddar [19]. Cheddar is an Ada 95 framework that provides tools and library to check whether AADL threads will meet their deadline at execution time. Cheddar uses the Ocarina libraries to analyze the AADL models reducing significantly the time of analysis of each of the prototypes modeled.

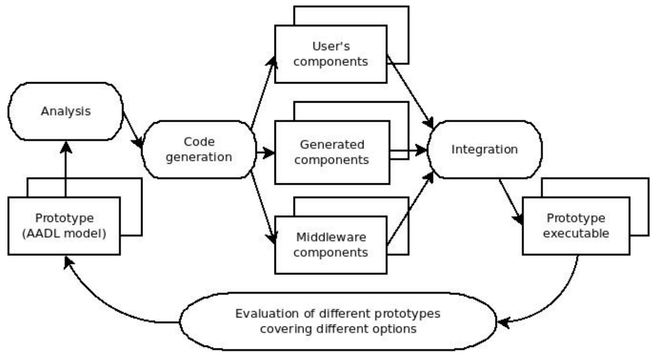

Fig. 3: Process for evaluating options through Throwaway prototypes

C. Executable Code Generation

We use code generation facilities in Ocarina to:

1) analyze the AADL model,
2) expand it, compute required resources and
3) generate code conforming to High-Integrity (HI) restrictions.

First, the prototype model is built by the application designer, he maps its application entities onto a hardware architecture. The throwaway prototype does not have the same hardware as the MC. The prototype works on an emulated environment (QEMU) [20]. Then, this architecture is tested for completeness and soundness, any mismatch in the configuration is reported by the analysis tool (e.g. lack of connection from an object). Consequently, model processing occurs, and the code is generated from this AADL model.

Code generation relies on well-known patterns for High-Integrity systems, supported by the minimal middleware called PolyORB-HI [21].

Figure 3 shows the entire process, in it can be seen as the developer and the customer participates in the refinement of the model and analysis of the options.

These steps allow the developer to go from the AADL model to executable code and forth, using one common model annotated with all required functional and non-functional elements, including its code base. Each tool works on the same model, allowing one to debug or enhance it at different steps.

Also, in our case study, this allowed that with small changes in the AADL model, different prototypes generated covering different options. Compared to a traditional process of upgrade of an OFP, it was observed that the incorporation of rapid prototyping with AADL has improved the quality and production of prototypes impacting positively on requirements definition process.

V. Conclusion

In this paper, we presented an application of modeling and rapid prototyping based on AADL in upgrading safety-critical systems such as an OFP, applied mainly at the stage of formulating and validating requirements.

Was described it as a process of updating an OFP, the stages that make up this process and particularly was introduced how the rapid prototyping techniques contribute to the definition stage both for the requirements engineers as well for customers too.

Rapid prototyping is a technique that has been used for many years in avionics systems but we selected the AADL to implement an efficient rapid prototyping process for upgrading the OFP, focusing on its design-by-refinement approach, and its extensibility through user-defined properties. We illustrated this approach by presenting the tool chain built around the Ocarina tool suite, Cheddar, QEMU and PolyORB-HI High-Integrity middleware.

We showed that an integrated set of tools enables the user to focus directly on system upgrades, and leverage its architecture to directly generate code for High-Integrity systems without any user intervention. The prototypes generated in this way can incorporate functional and nonfunctional requirements. Besides, analysis tools have been proposed to check model consistency and viability prior to generation. This increases confidence in the model while being fully automated.

Acknowledgment

The authors thank GSTR's (Real-Time System Group - Rio Cuarto National University) members for their feedback on earlier version of this work.

This work is partially funded by the PPI 18/B215 Project dependent on Science & Technical Secretary of the Rio Cuarto National University.

References

[1] D. Morris, “Avionics operational flight program software supportability,” Digital Avionics Systems Conference, 1990. Proceedings., IEEE/AIAA/NASA 9th , vol., no., pp.265,267, 15-18 Oct 1990.
[2] F. Kordon, Luqi : An introduction to Rapid System Prototyping, IEEE Transaction on Software Engineering Engineering, vol. 28 (9), pp. 817-821 (2002)
[3] F. Kordon, J. Henkel : An Overview of Rapid System Prototyping Today, Design Automation for Embedded Systems, vol. 8 (4), pp. 275-282 (2003).
[4] SAE. Architecture Analysis & Design Language (AS5506). available at http://www.sae.org, sep 2004.
[5] Modeling and Analysis of Real-time and Embedded systems. http://www.omg.org/omgmarte/
[6] ARINC 653: Avionics Application Software Standard Interface, Airlines Electronic Engineering Committee (AEEC). 1996.
[7] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff and F. Kordon. “Validate, simulate and implement ARINC653 systems using the AADL”. In ACM's 12th Annual International Conference on Ada and Related Technologies - SIGAda09.
[8] M. Masmano, Ismael Ripoll, A. Crespo, “An overview of the XtratuM nanokernel” 1st Intl. Workshop on Operating Systems Platforms for Embedded Real-Time applications. OSPERT 2005.
[9] M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll, A. Crespo and J.J. Metge, “LithOS: a ARINC-653 guest operating for XtratuM”. 12th Real-Time Linux Workshop. Nairobi. Kenya. 2010.
[10] T. Quatrami. “Visual Modeling with Rational Rose and UML”. Addison-Wesley, 2000.
[11] B. Kramer, Luqi, and V. Berzins, “Compositional semantics of a real-time prototyping language”, IEEE Transactions on Software Engineering, 19(5):453477, May 1993.
[12] Luqi, “Real-time constraints in a rapid prototyping language”, Computer Languages, 18(2):77103, 1993.
[13] D. Luckham, J. Vera, D. Bryan, L. Augustin, and F. Belz, “Partial orderings of event sets and their application to prototyping concurrent timed systems”, Journal of Systems Software, 21(3):253265, June 1993.
[14] D. Luckham, J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. “Specification and analysis of system architecture using Rapide”, IEEE Transactions on Software Engineering, 21(4):336354, April 1995.

[15] J. Hugues, B. Zalila, L. Paulet, F. Kordon, "Rapid Prototyping of Distributed Real-Time Embedded Systems Using the AADL and Ocarina", Rapid System Prototyping, IEEE International Workshop on, pp. 106-112, 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP '07), 2007.
[16] T. Vergnaud and B. Zalila, "Ocarina: a Compiler for the AADL", Technical report, Tlcom Paris, 2006. available at http://ocarina.enst.fr.
[17] M. Thompson, M. Heimdahl, "An integrated development environment for prototyping safety critical systems"; Proceedings of IEEE International Workshop on Rapid System Prototyping, Clearwater Beach, FL. 1999.
[18] Open Source AADL Tool Environment (OSATE). http://www.aadl.info.
[19] F. Singhoff, J. Legrand, L. N. Tchamnda, and L. Marc. "Cheddar : a Flexible Real Time Scheduling Framework." ACM Ada Letters journal, 24(4):1-8, ACM Press. Also published in the proceedings of the ACM SIGADA International Conference, Atlanta, 15-18 November, 2004, Nov. 2004.
[20] Quick EMUlator. http://www.qemu.org/
[21] J. Hugues, B. Zalila, and L. Pautet. "Middleware and Tool suite for High Integrity Systems". In Proceedings of RTSS-WiP06, Rio de Janeiro, Brazil, Dec 2006. IEEE.