

Platform Abstract Models for High-Integrity Real-Time Systems (Extended Abstract)

Richard Paige, Philippa Conmy, Alek Radjenovic,
Malcolm Wallace, and John McDermid
High-Integrity Systems Group,
Department of Computer Science, University of York, UK.
{paige, philippa, alek, malcolm, jam}@cs.york.ac.uk

Abstract

Ongoing work on the architecture-driven development of high-integrity real-time systems, and its relationship to the MDA initiative, is outlined. It is posited that separation between platform-independent and platform-specific models is not useful in this domain, due to certification needs, and suggestions as to how to reconcile MDA with the concerns of high-integrity real-time systems are outlined.

1 Introduction

The High-Integrity Systems Group is currently investigating the use of model-driven development in constructing high-integrity real-time systems (HIRTS). This work is under the aegis of the Defense and Aerospace Research Partnership, and involves participants from BAE Systems, Rolls-Royce, and QinetiQ. Of particular focus is the use of modelling for describing HIRTS that arise in the aerospace industry. Model-driven development is posited as useful for explicitly capturing architectural assumptions and design decisions that might otherwise be implicitly embedded in code. In general, architecture is often left implicit in much of the industrial development of HIRTS. Modelling is challenging to introduce in this domain because HIRTS typically have more intangible systems attributes – like safety criticality – than do other types of systems.

The main incentive to use model-driven development is to attempt to reduce the cost of managing incremental changes to requirements, in part by capturing implicit assumptions and design decisions, but also by explicitly modelling systems architecture. Incremental change is particularly difficult in the HIRTS domain, where certification of systems must take place. It is of increasing importance given the rapid changes in processing hardware. Often, hazard analyses and safety cases are not compositional, and a seemingly minor change to a system, its models, or its requirements will require that the entire system, or a suitably large subset of it, be re-certified. Late discovery of safety criticality is also a problem, as this has significant consequences for systems architecture. Dealing with these problems is expensive, so one of the aims of the project is to reduce the costs and impact of incremental change.

We have recently been examining the relationship of HIRTS models and modelling languages to the Model-Driven Architecture (MDA) initiative of the OMG. In particular, attention has been focusing on the separation between platform independent models (PIMs) and platform specific models (PSMs), which is a key element of the initiative designed to make systems more amenable to change. In this extended abstract, we briefly discuss the relationship between this separation and its application to HIRTS, and why we have reason to believe that it may be insufficient for dealing with incremental change in HIRTS models. We also suggest ways forward that we are examining for dealing with the incompatibility.

2 Models of HIRTS and Platform Independence

We are currently exploring the use of a combination of UML, Simulink, and Stateflow for modelling HIRTS architectures and behaviours. The reasons for this combination are both practical and historical. UML is a de facto standard for modelling, and techniques and advice exist for its use in architectural modelling. Simulink/Stateflow is widely used in the aerospace industry, and a suite of tools, e.g., Matlab and Reactis, exist. As well, the languages are reasonably complementary: UML is suitable for representing architectures, whereas Simulink/Stateflow is suitable for representing blocks, control/data flow, and behaviour. UML is being used for architectural modelling with reference to IEEE 1471 [2], in order to determine attributes that need to be modelled in an architecture. This is being augmented with feedback from the industrial partners, who are providing domain-specific information about attributes that need to be modelled in HIRTS architectures. We are also examining the Avionics Architectural Description Language (AADL) to see how it fits in with our work.

A key element of the systems that we are working with is that, in the past, separation between platform-specific information (e.g., the flight hardware that engine control software is to be installed upon) and platform-independent information is minimal: it is extremely difficult to isolate the platform-specific information, in part because of the need to carry out certification, and because of dependencies of HIRTS on their embedding systems, i.e., their operating environment. The reasons for such dependencies includes (a) the need to demonstrate that real-time performance can be achieved, and (b) assurance that the reliability of the underlying hardware is acceptable. For these reasons – and for reasons of certification – separation of models into PIMs and PSMs is seen as adding little of value. Instead, what is posited as useful is *platform abstract* models, discussed now in more detail.

3 Platform Abstract Models

Our initial analysis of MDA and HIRTS seems to suggest an immediate incompatibility: it is desirable to explicitly model the architecture of HIRTS, and the behaviour of components of functionality (e.g., control laws), but a separation between PIM and PSM does not seem possible, because of the embedded nature of the systems, and because of the need for certification. We discussed the former point in the previous

section; we now turn to the latter. In building safety critical systems, models are not certified as safe: it is the fully implemented system, including code, that is certified. Thus, the value provided by models comes from (a) providing abstraction facilities during the development process; (b) providing documentation for maintenance; (c) ideally providing structure for the certification (and re-certification) process; and (d) providing a basis for automated code generation. While separating PIM from PSM may provide assistance in (a), (b), and (d), it provides absolutely no assistance in managing certification, either incrementally or not.

Part of the difficulty of separating PIM from PSM in HIRTS is with the notion of independence: of what are the models independent? In a naive layered architecture for HIRTS, there is a suite of applications, which may make use of layers of drivers and APIs, a real-time operating system (RTOS), a bootloader, and underlying hardware. During development, we may need a suite of models which are independent of different aspects of this underlying infrastructure, i.e., at some times we may want a model that abstracts away from the hardware, at other times the RTOS, etc. When it is time to certify the system, we require a model that can be used to generate code that contains all the relevant details, and the code will then be certified.

Our preliminary investigations suggest that a separation between PIM and PSM for HIRTS is, in the general case, insufficient in order to effectively model the architecture of HIRTS, *and* to enable successful management of incremental change. Experience has also shown that attempting to mimic a separation of PIM and PSM in safety cases and hazard analysis is very difficult. What is needed instead is *platform abstract* models, wherein different levels of detail of the system are presented as suits the development tasks at hand. Platform abstract models are enabled by a number of tools and technologies, and the ones we are exploring are as follows.

1. An architectural modelling language, suitable for modelling HIRTS architectures, is needed. The language will likely integrate a profile of UML with Simulink/Stateflow, and may provide a migration path to using AADL as well.
2. Architectural models written in the languages chosen in (1) will be annotated with a number of pieces of textual information. Annotations would be applied to individual components and the entire model. Annotations will enable different views to be projected, and incremental certification to be carried out. The main types of annotations are:
 - *Contracts*, denoting rules on using component or system services. These can express behavioural information, or trace and temporal information (e.g., the component *always* provides a requested temperature in less than 20 milliseconds), or quality-of-service information. The types of contracts to be used hinges on the types of analysis required for certification.
 - *Safety contracts*, indicating hazards and faults that might arise throughout the architecture.
 - *Volatility tags*, indicating whether it is likely that the component will change, due to changes in requirements or environment.

- *Context*, specifying as precisely as is possible known facts and requirements about the environment for the component or system (e.g., bit depth, units), and what is required for the component or system to meet its requirements. The context will be vital in carrying out certification; in cooperation with volatility tags, incremental certification is more plausible.
3. A development process that has an iterative certification loop, supported by an infrastructure of static analysis and automated testing technology.

Platform abstract models have similarities with models of product families. These attempt to model features that can be reused across a suite of products. The main differences are that we are focusing on certification, we are treating platform abstraction as different views of a system, and tool support needs to be provided in order to project the different (consistent) views as needed during development.

4 Conclusions

We have argued that the separation of PSMs and PIMs, as suggested by the MDA initiative, is not entirely suitable for the development of high-integrity real-time systems. An architecture-driven development approach, supporting platform abstract models, based on the ideas of MDA, extended with context, volatility, and contract information – all aimed at improving the management of incremental certification – may be the way forward. We conclude by returning to the issue of views. In [5], it is said:

“... a view is a collection of models that represent one aspect of an entire system. A view applies to only one system, not to generalisations across many systems.”

It thus seems difficult to reason about views at the platform independent level. This is significant for HIRTS. In building and certifying HIRTS, there is a critical *safety view*, which will include elements like safety cases and determined hazards. This view persists from the start of development through implementation, certification, and deployment. Reasoning with this and other views used in modelling HIRTS is essential; they must be platform specific, according to the above. So platform independent models seem to provide little value for constructing high-integrity real-time systems.

References

- [1] C. Hofmeister et al. *Applied Software Architecture*, Addison-Wesley, 1999.
- [2] IEEE 1471 Recommended Requirements for Architectural Description, 2001.
- [3] B. Meyer. *Object-Oriented Software Construction*, Prentice-Hall, 1997.
- [4] Ministry of Defence. Requirements for Safety Related Software in Defence Equipment, DEF-STAN 00-55, 1997, UK.
- [5] OMG. *Model-Driven Architecture (MDA)*, Document ormsc/2001-07-01, July 2001. Available at www.omg.org.